

# THESIS REPORT

Master's Degree

## A Hybrid Control Strategy for Path Planning and Obstacle Avoidance with Non-holonomic Robots

*by V. Manikonda*

*Advisor: P.S. Krishnaprasad*

M.S. 94-8



*Sponsored by  
the National Science Foundation  
Engineering Research Center Program,  
the University of Maryland,  
Harvard University,  
and Industry*

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE <b>1994</b>		2. REPORT TYPE		3. DATES COVERED <b>00-00-1994 to 00-00-1994</b>	
4. TITLE AND SUBTITLE <b>A Hybrid Control Strategy for Path Planning and Obstacle Avoidance with Nonholonomic Robots</b>				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <b>University of Maryland, The Graduate School, 2123 Lee Building, College Park, MD, 20742</b>				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT <b>Approved for public release; distribution unlimited</b>					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT <b>see report</b>					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES <b>87</b>	19a. NAME OF RESPONSIBLE PERSON
a. REPORT <b>unclassified</b>	b. ABSTRACT <b>unclassified</b>	c. THIS PAGE <b>unclassified</b>			

# **A Hybrid Control Strategy for Path Planning and Obstacle Avoidance with Non-holonomic Robots**

by

Vikram Manikonda

Thesis submitted to the Faculty of the Graduate School  
of The University of Maryland in partial fulfillment  
of the requirements for the degree of  
Master of Science  
1994

Advisory Committee:

Professor P.S. Krishnaprasad, Chairman/Advisor  
Associate Professor J. Hendler  
Associate Professor W.P. Dayawansa



# **Abstract**

Title of Thesis: A Hybrid Control Strategy for Path Planning and Obstacle Avoidance with Non-holonomic Robots

Name of degree candidate: Vikram Manikonda

Degree and year: Master of Science, 1994

Thesis directed by: Professor P.S. Krishnaprasad  
Department of Electrical Engineering

Associate Professor J. Hendler  
Department of Computer Science

The primary focus is on providing a formal basis for behavior-based robotics using techniques that have been successful in control-based approaches for steering and stabilizing robots that are subject to nonholonomic constraints. In particular, behaviors for robots are formalized in terms of kinetic state machines, a motion description language and the interaction of the kinetic state machine with information coming in from (limited range) sensors. This allows us to create a mathematical basis for discussing these systems, including

techniques for integrating sets of behaviors. In addition we suggest optimality criteria for comparing both atomic and compound behaviors in various environments. A hybrid architecture for the implementation of path planners that use the motion description language is presented. The design and implementation of a planner for path planning and examples of obstacle avoidance with nonholonomic robots are discussed.

© Copyright by

Vikram Manikonda



# Acknowledgements

I would like to express my gratitude to my advisors Prof. P. S. Krishnaprasad and Prof. J. Hendler for their guidance, encouragement and for providing me with financial support during my graduate study. I would also like to thank Prof. W. P. Dayawansa for serving on my thesis committee.

Working with faculty and students from the departments of Electrical Engineering and Computer Science, provided me with an opportunity to develop a holistic view of the challenges in Robotics and I wish to thank my advisors and the Institute for Systems Research for having provided me with this unique opportunity. I would also like to thank Prof. R. W. Brockett whose work on Motion Description Languages served as a source of inspiration for the work in my thesis.

My colleagues in the PLUS group and the Intelligent Servo Systems Laboratory deserve a special mention for the many discussions on the subject and for making the office and the lab a wonderful place to work in.

I finally wish to thank my parents and my grandfather for their loving support and encouragement.

This research was supported in part by the National Science Foundation's Engineering Research Centers Program: NSFD CDR 8803012, and by the AFOSR University Research Initiative Program, under grant AFOSR-90-0105.

# Table of Contents

<u>Section</u>	<u>Page</u>
List of Figures	vi
1 Introduction	1
2 Controllability and Feedback Stabilization of Nonholonomic Systems	6
2.1 Nonholonomic Constraints . . . . .	7
2.2 Controllability of Nonholonomic Systems . . . . .	9
2.2.1 Equations of Motion for the Robotic Cart . . . . .	11
2.2.2 Equations of Motion for a Front-Wheel Drive Car . . .	12
2.3 Steering of Controllable Systems . . . . .	14
2.4 State Feedback Control . . . . .	15
3 Behavior Based Reactive Planning Systems	26
3.1 Kinetic State Machines . . . . .	27

3.2	Performance Measure of a Plan . . . . .	37
<b>4</b>	<b>Path Planning and Obstacle Avoidance</b>	<b>43</b>
4.1	Control Strategy . . . . .	44
4.2	Navigation Task Composition . . . . .	47
4.3	Partial Plan Generation . . . . .	51
4.3.1	Assumptions and Definitions . . . . .	52
4.3.2	Planning in the Obstacle Free Disk . . . . .	55
4.3.3	Tracing Boundaries of Obstacles . . . . .	57
4.4	Path Execution . . . . .	58
4.5	Learning and World Model Update Algorithm . . . . .	59
<b>5</b>	<b>Conclusions and Future Directions</b>	<b>67</b>
	<b>Bibliography</b>	<b>70</b>

# List of Figures

<u>Number</u>	<u>Page</u>
2.1 Lie bracket motion . . . . .	11
2.2 Robotic Cart . . . . .	12
2.3 Car-like Robot . . . . .	13
2.4 State Feedback Stabilization . . . . .	19
2.5 Trajectories of the Closed Loop System (2.15-2.16) . . . . .	22
2.6 Steering Algorithm . . . . .	24
2.7 Steering Algorithm . . . . .	25
3.1 Trajectory and Inputs Generated by the Plan . . . . .	35
3.2 Trajectory and Inputs Generated by the Plan . . . . .	37
3.3 Partial Plan Generation . . . . .	41
4.1 Hybrid Control Architecture . . . . .	45
4.2 Navigation Task Decomposition . . . . .	47
4.3 Representation of the World . . . . .	48

4.4	World Update . . . . .	49
4.5	Representation of a Plan . . . . .	50
4.6	Location and Calibration of IR sensors . . . . .	53
4.7	Paths Generated by the Planner . . . . .	60
4.8	Paths Generated by the Planner . . . . .	61
4.9	Learning and Plan Update Algorithm . . . . .	64
4.10	Plan Generated after Gaining Partial Knowledge of the World	65
4.11	Plan Generated after Gaining Partial Knowledge of the World	66

A Hybrid Control Strategy for Path Planning and Obstacle Avoidance with  
Nonholonomic Robots

Vikram Manikonda

August 12, 1994

**This comment page is not part of the dissertation.**

Typeset by  $\text{\LaTeX}$  using the `dissertation` style by Pablo A. Straub, University of

Maryland.



# Chapter 1

## Introduction

Traditional robot motion planning and obstacle avoidance concentrated on determining a path in the presence of holonomic or integrable, equality and inequality constraints on the configuration space. In the works of [1] dynamic path planning algorithms are suggested which use sensory information to generate paths assuming no *a priori* information on the size and shape of the obstacles. In [2] a graph search algorithm is presented to find obstacle free paths for a known environment.

On the other hand in the work of [3, 4, 5] we see a different approach to the holonomic motion planning problem using potential functions. The idea behind the approach in [5] was to associate signed charges with the goal, robot and obstacles and then generate a gradient field whose integral curves would steer the robot towards the goal which is constructed to be the global

minimum of the aggregate potential. Among the problems encountered in these approaches were those of undesirable local minima and cycles, when one considers intersecting obstacles.

To solve the problem of obstacle avoidance, in recent years we have seen significant research in AI in the areas of reactive control, one form of which is behavior based robotics. Unlike earlier systems which relied a great deal on knowledge representation, reactive systems rely on the direct coupling of sensory information and actuators. We see some examples of reactive planning systems in [6] where Brooks uses “task achieving behaviors” as the primary decomposition of the problem of autonomous navigation. He introduces the concept of subsumption architecture which is essentially a structured and layered set of behaviors with increasing levels of competence. Each layer is composed of augmented finite state machines. Other approaches along similar lines are seen in [7, 8] where a motor schema based approach is suggested for obstacle avoidance and path planning.

In practice however most robotic systems include constraints that are not holonomic. The kinematic constraints cannot be reduced to equivalent constraints on the configuration variables i.e. constraints cannot be integrated to give constraints which are explicit functions of position variables. A few examples of such systems are, models of a front wheel drive car, dextrous manipulation or assembly with robotic hands, attitude control of a satellite e.t.c.

Steering and stabilization of systems subject to nonholonomic constraints are being studied extensively. It is well known that these drift free completely nonholonomic systems, *where the number of controls is less than the number of states*, are controllable. Papers [9, 10, 11, 12] presented analytical tools based on Lie algebras to generate control sequences to steer these systems. As these nonholonomic, drift free systems do not satisfy Brockett’s necessary condition for smooth stabilization [13], these systems cannot be stabilized by using smooth time-invariant state feedback. This motivated the design of piecewise smooth feedback controllers [14], time-varying periodic controllers [15] and explicit control design to generate time-varying stabilizable control laws [16]. While most of the above research on steering and stabilization of nonholonomic systems assumes an obstacle-free world, we note that the problem of autonomous path planning and obstacle avoidance with nonholonomic robots is a nontrivial one. Traditional planners assume that arbitrary motion is permitted, and hence they cannot be applied to nonholonomic robots as they result in nonfeasible paths, i.e. trajectories that do not satisfy the constraints on the configuration variables. This motivates the need for a hybrid control strategy that integrates features of traditional planners and reactive systems with techniques that have been successful for control based approaches. In [17, 18, 19] possible solutions to solve this problem are presented.

In this thesis we present a path planner to solve the problem of real-time

obstacle avoidance and path planning with nonholonomic robots. Unlike earlier approaches the planner integrates features of reactive planning systems with modern control theory approaches to steer and stabilize nonholonomic robots. Planning is restricted to the two dimensional domain. The planner assumes that the robot has limited range sensors and information of the coordinates of the goal and its own coordinates at any instant of time. No restrictions are placed on the size and shape of the obstacles. As the first step towards the design of the planner we introduce a formal language for motion planning in which we model the robot as a kinetic state machine. The language enables us to define and reinterpret some of the existing notions of “behaviors”, “plans” etc. We then introduce a hybrid control strategy that is motivated by the hierarchical and distributed nature of neuromuscular control. Planning is done at two levels - global and local. For local planning obstacle free (non)feasible paths are generated using potential functions assuming that the robot is a point robot. A feasible path is then generated that obeys the constraints in the configuration variables. As feasible trajectories are only approximations to the trajectories generated using potential functions collision with obstacles while tracing the feasible trajectory is avoided by encoded sensor information in the kinetic state machine. In addition this information is also present in a lower level feedback loop while the robot is in motion. At a global level heuristics, along with the world map generated while

the robot is *en route* to the goal, are used to solve the problem of cycles encountered by using potential functions and also to improve the “performance” of the planner in situations where the same or similar tasks may have to be repeated.

The thesis is organized as follows. In chapter 2 we discuss some issues related to the controllability and steering of nonholonomic systems with specific reference to a cart with a two wheel drive and a car. We also review some of the recent work done in feedback stabilization of nonholonomic systems. In chapter 3 we introduce the notion of a kinetic state machine and attempt to formalize the notion of behaviors and plans for obstacle avoidance. In chapter 4 we present a new control architecture that integrates the features of behavior-based control systems and modern control theory. We also present an algorithm for obstacle avoidance and present some results from the simulations performed to test the algorithms. Details on the simulator are also provided. Chapter 5 is the conclusion and discusses some of the future work.

## Chapter 2

# Controllability and Feedback

## Stabilization of Nonholonomic Systems

A mobile robot, whose instantaneous configuration is completely determined by a set of generalized co-ordinates in the configuration space, is often subject to multiple constraints including geometric constraints and kinematic constraints. Kinematic constraints are expressed as relations between generalized coordinates and their time derivatives. If these constraints are explicitly integrable giving the form  $a_i(x) = k_i$  for some constant  $k_i$ , where  $x$  denotes generalized coordinates, then the motion of the system is restricted to a level surface of  $a_i$  and the constraints are called “holonomic”. In many mechanical systems it may not be possible to write the kinematic constraints as an

algebraic equality on the configuration space, i.e. the constraints may not be integrable. Such constraints are called “nonholonomic” and in this chapter we will review some properties related to the controllability and feedback stabilization [9, 10] of these nonholonomic systems. We assume that the reader is familiar with concepts such as manifolds and vectorfields (see [20] for a good exposition of these concepts).

## 2.1 Nonholonomic Constraints

Let us consider a mechanical system whose configuration space is an  $n$ -dimensional connected simply connected manifold  $\mathcal{M}$ . Let us represent each configuration of the system by an  $n$ -dimensional vector  $x = (x_1, x_2, \dots, x_n)$  where  $x_1, x_2, \dots, x_n$  are the generalized coordinates of the configuration space. Let the open set  $\Omega \subseteq \mathbb{R}^n$  be the set of all possible configurations of the system represented in the local-coordinates and let the motion of the system be represented by a smooth time function  $x(t)$ . The tangent vector to the trajectory at the point  $x$  is represented by  $\dot{x} = (\dot{x}_1(t), \dot{x}_2(t), \dots, \dot{x}_n(t))$  where  $\dot{x}_1(t), \dot{x}_2(t), \dots, \dot{x}_n(t)$  are the generalized velocities. We now consider two kinds of constraints that the system could be subject to.

1. *Classical Geometric Constraints* - These are analytical relations between the generalized coordinates. If the mechanical system is subject to  $m$  such

constraints, then there exists an  $m$ -dimensional vector function  $\Psi(x) : \Omega \rightarrow \mathbb{R}^m$  such that  $\Psi(x) = 0, \forall x \in \Omega$ . If the Jacobian matrix of  $\Psi(x)$  has full rank, then the constraints are independent and  $m$  generalized coordinates may be eliminated and  $n - m$  coordinates are sufficient to describe the system.

2. *Kinematic Constraints* - These constraints are represented by relations between generalized co-ordinates and their velocities, written in matrix form as

$$A^T(x)\dot{x} = 0 \quad (2.1)$$

where

$$A(x) = (a_1(x), a_2(x), \dots, a_k(x))$$

and  $a_1^T, \dots, a_k^T$  are smooth  $n$ -dimensional covector fields on  $\mathcal{M}$ . In the rest of the discussion we assume that associated geometric constraints have been eliminated and (2.1) represents the system with  $m$  independent constraints.

Let  $\Delta$  be a smooth nonsingular distribution that annihilates the codistribution given by the covector fields  $a_1^T, \dots, a_k^T$  i.e.

$$\Delta = \text{span}\{b_1, b_2, \dots, b_m\}, \quad m = n - k \quad (2.2)$$

where  $b_1, b_2, \dots, b_m$  are a set of  $m = n - k$  smooth vector fields, that satisfy the relation

$$a_j^T(x)b_i(x) = 0, \quad \forall x \in \Omega, \quad j = 1, \dots, k, \quad i = 1, \dots, m.$$

Let  $B(x)$  be the full rank matrix made up of the vector function  $b_i(x)$  i.e.  $B(x) = (b_1(x), \dots, b_m(x))$ . Now (2.1) may be expressed as

$$\dot{x} \in \Delta(x) \text{ or } \dot{x} \in \text{Im}(B(x)) \quad (2.3)$$

Along the trajectories of the system, equation (2.3) implies the existence of a vector time function  $u(t) \in \mathbb{R}^m$  for all  $t$  such that

$$\dot{x} = B(x)u(t) \quad (2.4)$$

where  $B(x)$  is the full rank matrix as defined above. Equation (2.4) implies that for any initial condition  $x(0)$  and any time function  $u(t)$  the solution of  $x(t)$  of (2.4) will satisfy (2.1).

Hence the system as defined by (2.4) can be viewed as a control system with an  $n$ -dimensional state space representation of a nonholonomic system with state  $x(t)$  and control  $u(t)$ . Systems of the form (2.4) are referred to as drift free systems and have some nice properties due to their symmetry, e.g. any trajectory can be followed in either direction by changing the sign of the control.

## 2.2 Controllability of Nonholonomic Systems

We now address the issue of controllability of systems defined by (2.4) i.e.

$$\dot{x} = b_1(x)u_1 + \dots + b_m(x)u_m \quad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m.$$

We are interested in determining conditions for the existence of controls  $u : [0, T] \rightarrow \mathbb{R}^m$ ,  $T \geq 0$  that steer the system from any given initial state  $x_o = x(0)$  to  $x_f = x(T)$  where  $x_o, x_f \in \Omega$ . Controllability of (2.4) can be characterized in terms of the Lie algebra generated by the vector fields  $b_i$ . The Lie bracket of two vector fields  $b_i$  and  $b_j$  is defined as

$$[b_i, b_j] = \frac{\partial b_j}{\partial x} b_i - \frac{\partial b_i}{\partial x} b_j \quad (2.5)$$

We now define  $\Delta^*$  as the involutive closure of the distribution of  $\Delta$  (as defined in (2.2)) under Lie bracketing. Then the conditions for controllability are given by Chow's theorem.

**Theorem 1 (Chow)** [10] *If  $\Delta^*(x) = \mathbb{R}^n$  for each  $x \in \Omega$  then the system (2.4) is controllable.*

A useful interpretation of Chow's theorem can be obtained by using the following characterization of the Lie bracket. Let  $\phi_t^f : \Omega \subset \mathbb{R}^n \mapsto \mathbb{R}^n$  denote the flow of a vector field for a time  $t$ . Consider the sequence of flows shown in the Fig 2.1. The net motion satisfies

$$\phi_\epsilon^{-g} \circ \phi_\epsilon^{-f} \circ \phi_\epsilon^g \circ \phi_\epsilon^f(x_0) = \epsilon^2[f, g] + o(\epsilon^2)$$

In other words if we can move in every direction using Lie bracket motion the system is controllable.

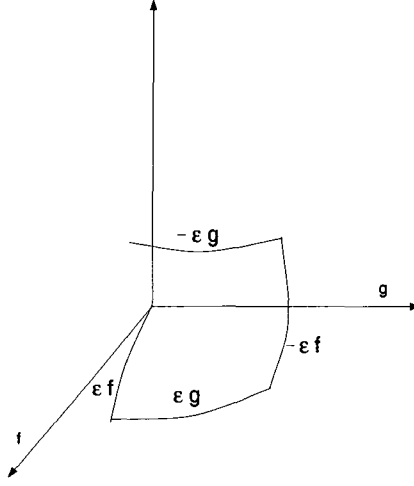


Figure 2.1: Lie bracket motion

### 2.2.1 Equations of Motion for the Robotic Cart

Let  $(x, y) \in \mathbb{R}^2$  denote the position of the robot w.r.t. some inertial frame and  $\theta \in S^1$  (see Fig. 2.2) denote the orientation of the robot relative to the  $x$ -axis. Hence the configuration space of the robot is  $Q = \mathbb{R}^2 \times S^1$ . Let  $\omega_1, \omega_2 \in \mathbb{R}$  denote the angular velocities of the wheels. Hence the kinematic equations of the cart are

$$\begin{aligned} v_x &= \dot{x} = \frac{r}{2}(\omega_1 + \omega_2) \cos \theta = u_1 \cos \theta \\ v_y &= \dot{y} = \frac{r}{2}(\omega_1 + \omega_2) \sin \theta = u_1 \sin \theta \\ \dot{\theta} &= \frac{r}{b}(\omega_1 - \omega_2) = u_2 \end{aligned} \tag{2.6}$$

where

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{b} & -\frac{r}{b} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

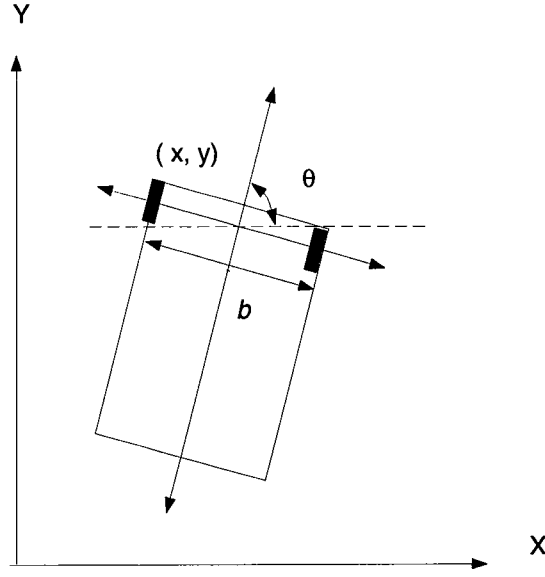


Figure 2.2: Robotic Cart

Rewriting (2.6) as

$$\Sigma_{cart} : \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{pmatrix} u_1 + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u_2$$

This system  $\Sigma_{cart}$  is a drift free system and is of the form (2.4). We see that

$\{b_1, b_2, [b_1, b_2]\}$  spans  $\mathfrak{R}^3$  and hence the system is controllable.

## 2.2.2 Equations of Motion for a Front-Wheel Drive

### Car

Let  $(x, y) \in \mathfrak{R}^2$  denote the position of the front wheel drive car w.r.t. to some inertial frame,  $\theta \in S^1$  denote the orientation of the car relative to the  $x$  axis

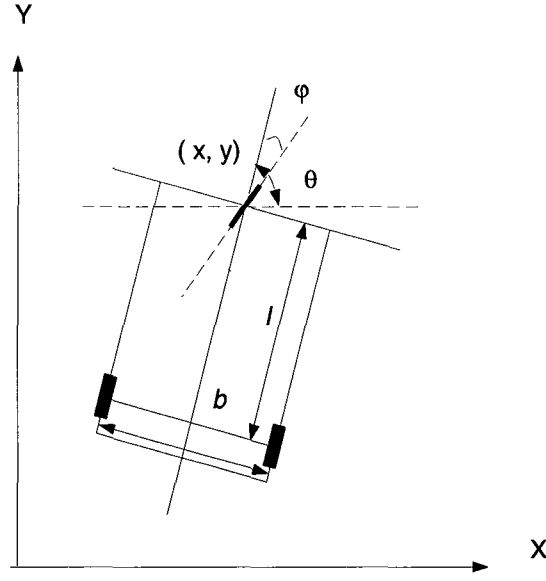


Figure 2.3: Car-like Robot

and let  $\phi \in S^1$  denote the steering angle (see Fig 2.3). Hence the configuration space of the car is  $Q = \mathbb{R}^2 \times S^1 \times S^1$ . The constraints for the front and rear wheels are

$$\dot{x} \sin(\theta - \phi) - \dot{y} \cos(\theta - \phi) = 0$$

$$\frac{d}{dt}(x - l \cos \theta) \sin \theta - \frac{d}{dt}(y - l \sin \theta) \cos \theta = 0$$

Let  $v \in \mathbb{R}$  the heading velocity, and  $\dot{\phi} \in \mathbb{R}$  the steering velocity, be the inputs to the system. The kinematic equations of the car written in the state variable

form are

$$\Sigma_{car} : \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos(\theta - \phi) \\ \sin(\theta - \phi) \\ \frac{-1}{l} \sin \phi \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \dot{\phi} \quad (2.7)$$

This system  $\Sigma_{car}$  is a drift free system and is of the form (2.4). We see that  $\{b_1, b_2, [b_1, b_2], [b_1, [b_1, b_2]]\}$  spans  $\mathbb{R}^4$  and hence the system is controllable .

## 2.3 Steering of Controllable Systems

One of the solutions suggested to steer these systems is driving them by periodic functions [10]. For example consider the system  $\Sigma_{cart}$  that was discussed in the previous section. Note that this system is exactly the same as the unicycle [12]. Now once again modifying the input to

$$v_1 = u_1 \cos \theta$$

$$v_2 = u_2$$

and relabeling the states ( $x_1 = x$ ,  $x_2 = \theta$ ,  $x_3 = y$ ) we have

$$\begin{aligned} \dot{x}_1 &= v_1 \\ \dot{x}_2 &= v_2 \\ \dot{x}_3 &= v_2 \tan x_2 \end{aligned} \quad (2.8)$$

Since the states  $x_1$  and  $x_2$  are independently controlled by inputs  $v_1$  and  $v_2$  we can steer  $x_1$  and  $x_2$  to the desired values. This may cause  $x_3$  to drift. To steer

$x_3$  to its desired value we choose  $v_1 = \alpha \sin(\omega t)$  and  $v_2 = \beta \cos(\omega t)$ . Solving for  $x_1$  we have

$$\dot{x}_3 = \tan\left(\frac{\beta}{\omega} \sin \theta\right) \alpha \sin \theta d\theta \quad (2.9)$$

Observe that the value of  $x_3$  after  $\frac{2\pi}{\omega}$  advances by

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} \tan\left(\frac{\beta}{\omega} \sin \theta\right) \alpha \sin \theta d\theta$$

Hence  $\alpha$ ,  $\beta$ , and  $\omega$  can be selected appropriately to steer  $x_3$  to the desired value.

## 2.4 State Feedback Control

In this section we discuss the existence of smooth pure state feedback control to stabilize of the form (2.4). The discussion is essentially a review of some aspects of papers [9, 13].

A pure state feedback control law is defined as a smooth mapping

$$u : \Omega \rightarrow \mathbb{R}^m : u \rightarrow u(x)$$

with the additional condition that  $u(0) = 0$ . The application of this feedback law to (2.4) results in closed loop dynamics of the form

$$\dot{x} = B(x)u(x)$$

which has the origin as the equilibrium point.

From the previous discussion of controllability of (2.4) we know that there exists a control law that will drive the system to the origin, but this does not ensure the existence of a smooth pure state feedback law that will asymptotically stabilize the origin. In fact there exists no pure state feedback law that asymptotically stabilizes the origin. Before we state a proof to show that there exists no pure state feedback to stabilize the origin we state (without proof) the necessary condition for feedback stabilization given by Brockett [13].

**Theorem 2 (Brockett)** *Let  $\dot{x} = f(x, u)$  be given with  $f(x_0, 0) = 0$  and  $f(., .)$  continuously differentiable in a neighborhood of  $(x_0, 0)$ . A necessary condition for the existence of a continuously differentiable control law that makes  $(x_0, 0)$  asymptotically stable is that:*

*(i) the linearized system should have no uncontrollable modes associated with eigenvalues whose real parts are positive.*

*(ii) there exists a neighborhood  $N$  of  $(x_0, 0)$  such that for each  $\xi \in N$  there exists a control  $u_\xi(.)$  defined on  $[0, \infty)$  such that this control steers the solution of  $\dot{x} = f(x, u_\xi)$  from  $x = \xi$  at  $t = 0$  to  $x = x_0$  at  $t = \infty$ .*

*(iii) the mapping  $\gamma : \Omega \times \mathbb{R}^m \rightarrow \mathbb{R}^n$  defined by  $\gamma : (x, u) \rightarrow f(x, u)$  should be onto an open set containing 0.*

**Corollary 1** *Given a nonholonomic system of the form (2.4) there does not*

exist a smooth pure state feedback law of the form  $u_1 = u(x) \cdots u_m = u(x)$ , that makes the equilibrium point  $x = 0$  asymptotically stable.

**Proof:** From the smoothness in  $A(x)$ , as defined in (2.1), and the independence of the constraints, we know that there exists a neighborhood  $\mathcal{V}_0$  of the origin in  $\mathbb{R}^n$ , such that a given a set of  $m$  rows of  $A(x)$  are independent on  $\mathcal{V}_0$ . Without loss of generality, we assume that the first  $m$  rows of  $A(x)$  are independent on  $\mathcal{V}_0$ , and we partition  $A(x)$  as follows

$$A(x) = \begin{pmatrix} A_1(x) \\ A_2(x) \end{pmatrix} \quad (2.10)$$

where  $A_1(x)$  is a square matrix on  $\mathcal{V}_0$ .

Let  $\mathcal{V}_1$  be a neighborhood in  $\mathbb{R}^m$ , containing the origin, and let  $\mathcal{V}$  be the cartesian product of  $\mathcal{V}_0$  and  $\mathcal{V}_1$ . Consider the mapping

$$(x, u) \rightarrow g(x, u) = B(x)u \quad (2.11)$$

and denote as  $\mathcal{W}$ , the image of  $\mathcal{V}$  by this mapping  $g$ . The for any  $\sigma$  belonging to  $\mathcal{W}$ ,  $\exists x$  such that  $\sigma \in \text{Im}(B(x))$  and therefore

$$A^T_1(x)\sigma_1 + A^T_2(x)\sigma_2 = 0 \quad (2.12)$$

where  $\sigma$  is partitioned in a  $m$ -subvector  $\sigma_1$  and a  $(n - m)$ -subvector  $\sigma_2$ . This implies that any  $\sigma$ , with  $\sigma_1$  not equal to zero and  $\sigma_2$  equal to zero, does not belong to  $\mathcal{W}$ , and therefore that  $\mathcal{W}$ , the image of the open set  $\mathcal{V}$ , is not in the

open neighborhood of the origin. The result then follows from *Theorem 2(iii)*

(Brockett's necessary condition for the existence of smooth feedback)  $\square$

**Theorem 3** [9] *With the smooth feedback control law*

$$u(x) = -B^T(x)x$$

*the equilibrium point  $x = 0$  of the closed loop system is globally marginally stable. Precisely:*

*a) the state  $x(t)$  is bounded as follows for all  $t$  :  $\|x(t)\| \leq \|x(0)\|$*

*b) the state  $x(t)$  converges to the invariant set  $U$ :*

$$U = \{x \mid -B^T(x)x = 0\}$$

**Proof.** Consider the Lyapunov function  $V(x) = xx^T$

$$\dot{V}(x) = -2x^T B(x)B^T(x)x \leq 0$$

along the trajectories of the closed loop system.  $\square$

**Example:** Consider the robotic cart given by (2.6). The feedback law

$$u_1 = -x \cos \theta - y \sin \theta$$

$$u_2 = -\theta$$

stabilizes the system to the invariant set  $U = \{x = 0, y = k \in \mathfrak{R}, \theta = 0\}$ .

Fig 2.4 shows the trajectories of the system with the above feedback law.

In path planning problems in many situations, e.g. wall tracing, we are interested in case where the system converges to trajectories parallel to the

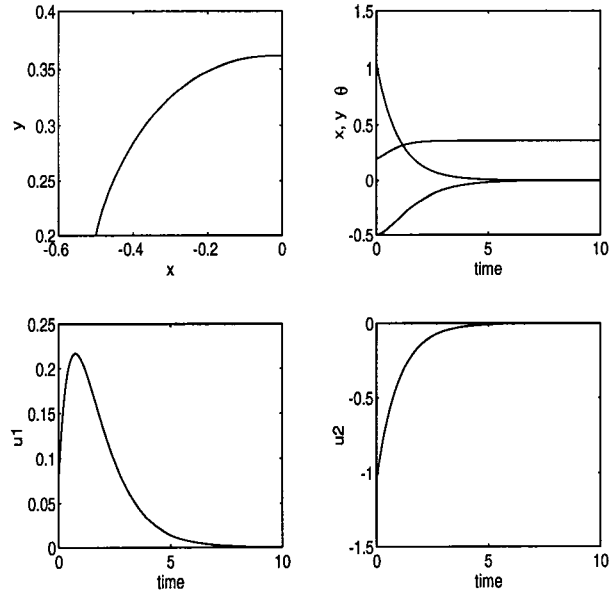


Figure 2.4: State Feedback Stabilization

the boundaries of the obstacles. With an appropriate change of coordinates in the case of a car-like robot, this problem reduces to finding a control law that drives the system to the set  $U = \{x_1 \in \mathbb{R}, x_2 = 0, x_3 = 0, x_4 = 0\}$ . We suggest a control law and an algorithm to steer a car like robot (2.13 - 2.16) (which is simply (2.7) with new variables) to the invariant set  $U$ . Before we design the control law consider consider the system described by (2.15) and (2.16)

$$\dot{x}_1 = \cos(x_3 - x_4)u_1 \quad (2.13)$$

$$\dot{x}_2 = \sin(x_3 - x_4)u_1 \quad (2.14)$$

$$\dot{x}_3 = -\frac{\sin(x_4)}{l}u_1 \quad (2.15)$$

$$\dot{x}_4 = u_2 \quad (2.16)$$

**Proposition 1** *With the smooth feedback law:*

$$u_1 = k_1, \quad u_2 = k_2 \sin(x_3 - x_4), \quad k_1, k_2 \in \mathfrak{R}^+ \quad (2.17)$$

*the origin of closed loop system of (2.15), (2.16) is asymptotically stable.*

**Proof:** Let  $D = \{-\pi \leq x_3 \leq \pi, -\pi/2 < -\phi_{min} \leq x_4 \leq \phi_{max} < +\pi/2\}$ .

Consider the function  $V : D \rightarrow \mathfrak{R}$  defined by

$$V(x) = a \int_0^{x_4} \sin y dy + b \sin^2 \frac{(x_3 - x_4)}{2}$$

Along trajectories of the closed loop system

$$\begin{aligned} \dot{V}(x) &= a \sin(x_4) \dot{x}_4 + 2b \sin \frac{(x_3 - x_4)}{2} \cos \frac{(x_3 - x_4)}{2} \frac{1}{2} (\dot{x}_3 - \dot{x}_4) \\ &= ak_2 \sin(x_4) \sin(x_3 - x_4) + \frac{b}{2} \sin(x_3 - x_4) \left(-\frac{k_1}{l} \sin(x_4) - k_2 \sin(x_3 - x_4)\right) \\ &= \left(ak_2 - \frac{b}{2l}k_1\right) \sin(x_4) \sin(x_3 - x_4) - \frac{b}{2}k_2 \sin^2(x_3 - x_4) \end{aligned}$$

Choose  $a = \frac{k_1}{2l}$ ;  $b = k_2$  then

$$\begin{aligned} \dot{V}(x) &= \frac{-k_2^2}{2} \sin^2(x_3 - x_4) \\ &\leq 0 \end{aligned}$$

Let  $S$  be the set of points where  $\dot{V}(x) = 0$ . Hence  $S = \{x_3, x_4 | x_3 - x_4 = 0\}$ .

Let us assume that  $x(t)$  is a trajectory that belongs to  $S$  for all  $t$ .

$$\Rightarrow \dot{x}_3 - \dot{x}_4 = -\frac{k_1}{l} \sin(x_4) - k_2 \sin(x_3 - x_4) = 0$$

$$\Rightarrow x_4 = 0, \Rightarrow x_3 = 0$$

Thus the only solution that can stay in  $S$  is the trivial solution and hence it follows from La Salle's Invariance Principle that the origin is asymptotically stable.  $\square$

Since  $(x_3, x_4) = (0, 0)$  is an asymptotically stable equilibrium point, by continuity there exists a ball  $B_\delta = \{x_3, x_4 \in S | V(x_3, x_4) < \delta\}$  such that  $\|x_3 - x_4\|$  is small and the system can be approximated by

$$\dot{x}_3 = -k_1 x_4 \tag{2.18}$$

$$\dot{x}_4 = k_2 (x_3 - x_4)$$

Written as a second order system is of the form

$$\ddot{y} + k_2 \dot{y} + \frac{k_2}{k_1} y = 0 \tag{2.19}$$

which is the equation of a damped oscillator. Hence we can choose  $k_1$  and  $k_2$  and obtain a desired rate of convergence. Note that once in  $B_\delta$ ,  $k_2$  determines the rate of convergence, but  $k_1$  determines the rate at which the states enter  $B_\delta$ . Fig. 2.5 show the plots of the inputs and states for the same initial conditions and different values of  $k_1$  and  $k_2$ .

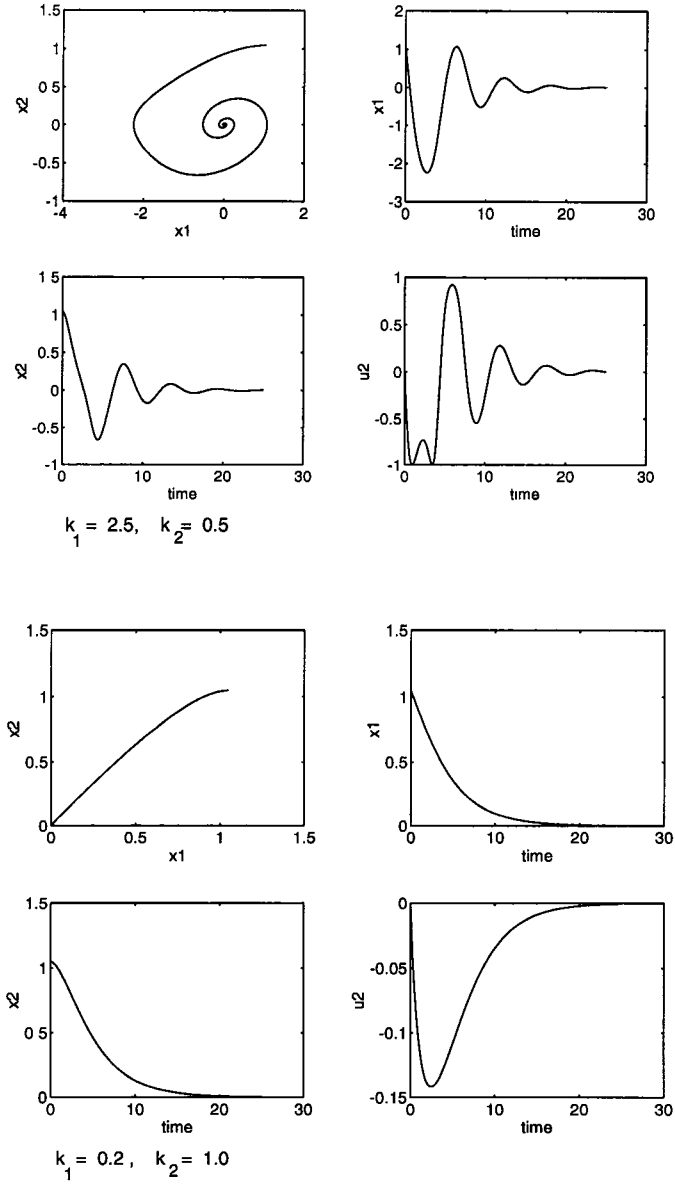


Figure 2.5: Trajectories of the Closed Loop System (2.15-2.16)

Now observe that with the given feedback law (2.14) and (2.16) have the same RHS and hence

$$x_2(t) = x_2(0) + \frac{k_1}{k_2}x_4(t) - \frac{k_1}{k_2}x_4(0) \quad (2.20)$$

Since the closed loop system (2.15 - 2.16) is asymptotically stable,

$$\lim_{t \rightarrow \infty} x_4(t) = 0$$

hence,

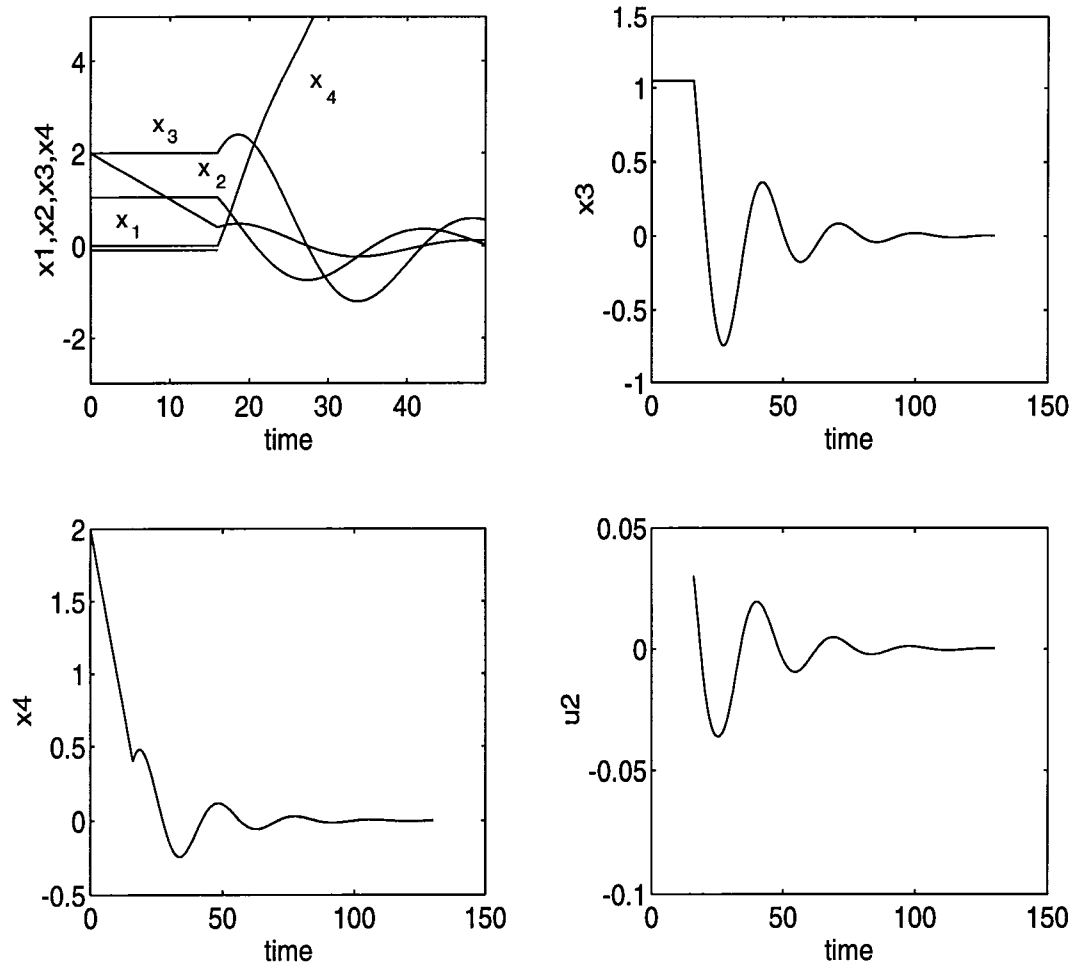
$$\lim_{t \rightarrow \infty} x_2(t) = x_2(0) - \frac{k_1}{k_2}x_4(0) \quad (2.21)$$

Observing that the state  $x_4$  can be steered directly by control  $u_2$  and we have the freedom to choose the gains  $k_1$  and  $k_2$ , we can steer the car to the invariant set  $U = \{x_1 \in \mathfrak{R}, x_2 = 0, x_3 = 0, x_4 = 0\}$ .

### Steering algorithm

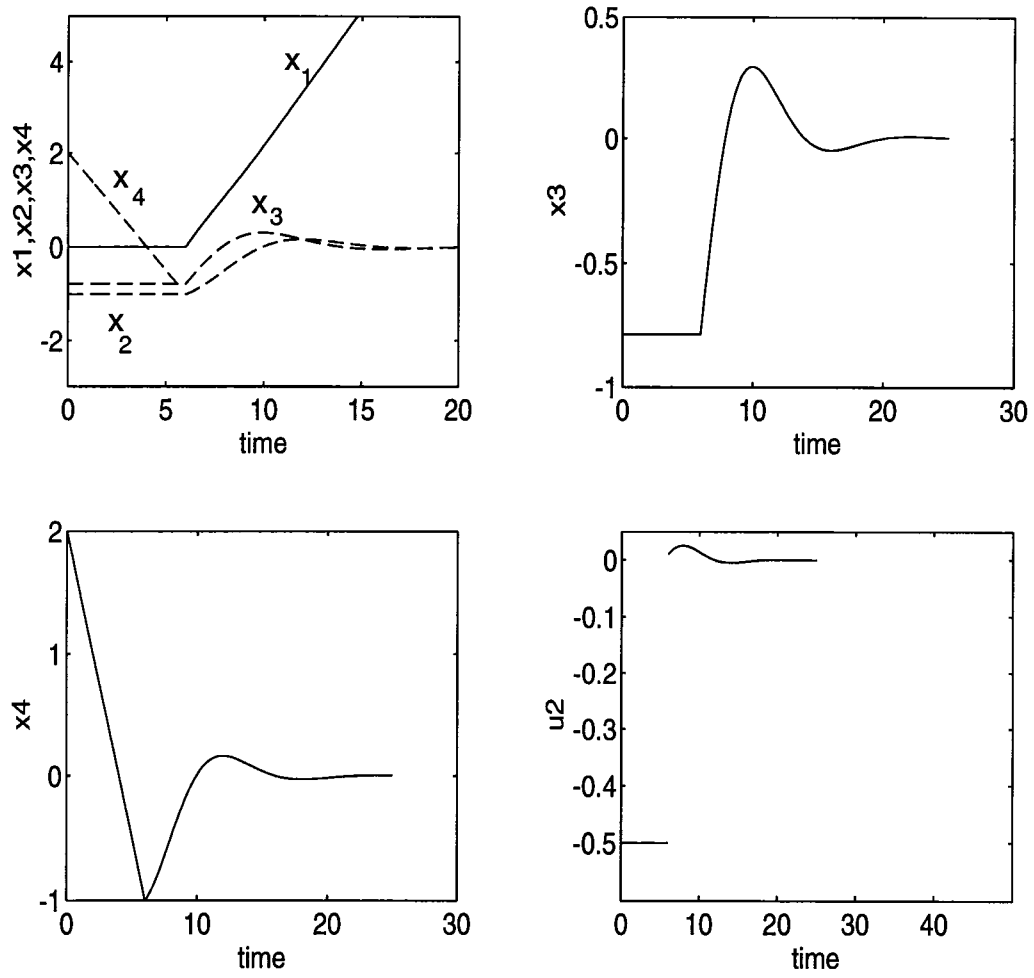
- (i). Depending on the initial condition  $x_2(0)$  and  $x_4(0)$ , and desired convergence rates choose gains  $k_1$  and  $k_2$ .
- (ii). Apply the control law  $u_1 = 0, u_2 = c, c \in \mathfrak{R}$  for time  $t = (x_2(0)\frac{k_2}{k_1} - x_4(0))/c$ .
- (iii). Apply the control law (2.17).

Observe that since we have the freedom to choose the gains in practical applications the closed loop control can be turned off in some *finite time*. Fig. 2.6 and Fig 2.7 show the application of the control algorithm.



$$x_1 = 0, x_2 = 2, x_3 = \pi/3, x_4 = 2.0, k_1 = 0.5, k_2 = 0.1$$

Figure 2.6: Steering Algorithm



$$x_1(0) = 0, x_2(0) = -1, x_3(0) = -\pi/4, x_4(0) = 2, k_1 = 0.6, k_2 = 0.6, c = -0.5$$

Figure 2.7: Steering Algorithm

## Chapter 3

# Behavior Based Reactive Planning Systems

In chapter 2 we discussed some control laws for steering and stabilizing drift free nonholonomic systems. The control laws discussed assumed that no obstacles are present. Modeling obstacles as constraints in the configuration space and then designing control laws can be a fairly complex problem. Earlier approaches used simple open loop control laws or “behaviors” such as *move* or *turn* in a reactive way to steer these robots. Unlike earlier systems which relied a great deal on knowledge representation, reactive systems, rely on the direct coupling of sensory information and actuators. We see some examples of reactive planning systems in [6] where Brooks uses “task achieving behaviors” as the primary decomposition of the problem of autonomous navigation. He introduces the concept of a subsumption architecture which is

essentially a structured and layered set of behaviors with increasing levels of competence. Each layer is characterized by *augmented finite state machines* [6]. But having had a better understanding of the properties of nonholonomic systems, as in chapter 2, one would like to exploit the underlying geometry along with real time sensor information for path planning and obstacle avoidance. Hence there is a need for a language that can capture and integrate features of modern control theory and reactive planning systems. Motivated by [21, 22] in this chapter we describe a formal language for path planning and obstacle avoidance. The language also gives us the means to formalize, concepts such as “behavior”, “plan” etc. used in the path planning literature.

### 3.1 Kinetic State Machines

We treat a robot as a kinetic state machine [21] which can be thought of as a continuous analog of a finite automaton. Kinetic state machines are governed by differential equations of the form

$$\dot{x} = \sum_{i=1}^m b_i(x)u_i; \quad y = h(x) \in \mathbb{R}^p \quad (3.1)$$

where

$$x(\cdot) : [0, T] \rightarrow \mathbb{R}^n$$

$$u_i : \mathbb{R}^+ \times \mathbb{R}^p \rightarrow \mathbb{R}$$

$$(t, y(t)) \mapsto u_i(t, y(t))$$

Further  $b_i$  is a vector field in  $\mathbb{R}^n$ .

We now define the atoms of the motion language as triples of the form  $(U, \xi, T)$  where

$$U = (u_1, \dots, u_m)^T$$

where  $u_i$  is as defined earlier,

$$\xi : \mathbb{R}^k \rightarrow \{0, 1\}$$

$$s(t) \mapsto \xi(s(t))$$

is a boolean function,  $T \in \mathbb{R}^+$  and  $s(\cdot) : [0, T] \rightarrow \mathbb{R}^k$  is a  $k$  dimensional signal that represents the state of the  $k$  sensors and  $\xi$  can be interpreted as an interrupt to the system which is activated in a case of emergency, e.g. the robot crashes into an obstacle, or gets too close to an obstacle. Let us denote  $\hat{t}$ ,  $0 \leq \hat{t} \leq T$  as the time at which the interrupt was received i.e.  $\xi$  changes state from 1 to 0.

If at time  $t_0$  the kinetic state machine receives an input atom  $(U, \xi, T)$  the state will evolve governed by the differential equation (3.1), as

$$\dot{x} = B(x)U, \quad \forall t, t_0 \leq t \leq t_0 + \min[\hat{t}, T]$$

. If the kinetic state machine receives an input string  $(U_1, \xi_1, T_1) \cdots (U_n, \xi_n, T_n)$

then the state  $x$  will evolve according to

$$\begin{aligned}
\dot{x} &= B(x)U_1, \quad t_0 \leq t \leq t_0 + \min[\widehat{t}_1, T_1]. \\
&\vdots \\
\dot{x} &= B(x)U_n, \quad t_0 + \min[\widehat{t}_1, T_1] + \cdots + \min[\widehat{t}_{n-1}, T_{n-1}] \\
&\leq t \leq t_0 + \cdots + \min[\widehat{t}_n, T_n].
\end{aligned} \tag{3.2}$$

Hence we may denote a kinetic state machine as a seven-tuple  $(\mathcal{U}, \mathcal{X}, \mathcal{Y}, \mathcal{S}, B, h, \xi)$ ,

where

$\mathcal{U} = (C^\infty(\mathbb{R}^+ \times \mathbb{R}^p); \mathbb{R}^m)$  is an input (control) space,

$\mathcal{X} = \mathbb{R}^n$  is the state space,

$\mathcal{Y} = \mathbb{R}^p$  is an output space,

$\mathcal{S} \subset \mathbb{R}^k$  is the sensor signal space,

$B$  is an  $\mathbb{R}^{n \times m}$  matrix (constraints matrix),

$h : \mathcal{X} \rightarrow \mathcal{Y}$  maps the state space to the output space and

$\xi : \mathcal{S} \rightarrow \{0, 1\}$  maps the sensor signal to the set  $\{0, 1\}$ .

**Definition:** Given an atom,  $(U, \xi, T)$ , define  $(\alpha U, \xi, \beta T)$ ,  $\alpha \in \mathbb{R}, \beta \in \mathbb{R}^+$  as the corresponding *scaled atom* and denote it as  $(\alpha, \beta)(U, \xi, T)$ .

**Definition:** An *alphabet*  $\Sigma$  is a finite set of atoms, i.e  $(U, \xi, T)$  triples.

Thus  $\Sigma = \{(U_1, \xi_1, t_1), \dots, (U_n, \xi_n, t_n)\}$  for some finite  $n$  or equivalently  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$  where  $\sigma_i$  denotes the triple  $(U_i, \xi_i, T_i)$ , such that  $\sigma_i \neq (\alpha, \beta)(\sigma_j)$   $\alpha \in \mathbb{R}, \beta \in \mathbb{R}^+$  and  $i = 1, \dots, n, j = 1 \dots n$ .

**Definition:** An *extended alphabet*  $\Sigma_e$  is the set of scaled atoms, i.e  $(\alpha U, \xi, \beta T)$  triples derived from the alphabet  $\Sigma$ .

**Definition:** A *language*  $\Sigma^*$  ( $\Sigma_e^*$ ) is defined as the set of all strings over the fixed alphabet  $\Sigma$  (extended alphabet  $\Sigma_e$ ).

**Definition:** A *behavior*  $\pi$  is an element of the extended language  $\Sigma_e^*$ . For example given an alphabet  $\Sigma = \{\sigma_1, \sigma_2\}$  a behavior  $\pi$  could be the string  $(\alpha_1, \beta_1)\sigma_1(\alpha_2, \beta_2)\sigma_2(\alpha_3, \beta_3)\sigma_1$ .

**Remark:** To account for constraints one might limit behaviors to lie in a sublanguage  $B \subset \Sigma_e^*$ . This will be explored in future work.

To simplify notation we denote the scaled atom  $(1, 1)\sigma_i$  simply by  $\sigma_i$ .

**Definition:** The *length* of a behavior denoted by  $|\pi|$  is the number of atoms (or scaled atoms) in the behavior.

**Definition:** The *duration*  $T(\pi)$  of a behavior

$$\pi = (\alpha_1, \beta_1)(U_1, \xi_1, T_1) \cdots (\alpha_n, \beta_n)(U_n, \xi_n, T_n)$$

executed beginning at time  $t_0$  is the sum of the time intervals for which each of the atoms in the behavior was executed. That is,

$$T(\pi) = t_0 + \min[\hat{t}_1, \beta_1 T_1] \cdots + \min[\hat{t}_n, \beta_n T_n] \quad (3.3)$$

**Definition :** Given a kinetic state machine and a world-model, a *plan*  $\Gamma$  is defined as an ordered sequence of behaviors, which when executed is guaranteed to achieve the given goal. For example a plan  $\Gamma = \{\pi_3 \pi_1 \pi_n \cdots\}$  could be

generated from a given language where each behavior is executed in the order in which they appear in the plan. The length of a plan,  $|\Gamma| = \sum_{i=1}^n |\pi_i|$ . The duration of the plan is given by  $T(\Gamma) = \sum_{i=1}^n T(\pi_i)$ . In a particular context there may be more than one plan that achieves a given goal.

**Remark:** Since each atom when executed by a kinetic state machine, combines in general both open-loop and feedback controls, one could argue that our definition of behavior captures the essence of behavior (say in movement) in biology, as well as the sense in which the term is used by Brooks [6].

Given a nonholonomic robot, an environment and a certain task, some important questions that arise are -

(i) How does one choose an alphabet  $\Sigma$ , or even a more elementary question - does there exist a  $\Sigma$  which can be used to generate behaviors and hence plans to achieve the required goal ?

(ii) Given an alphabet set of behaviors is infinite. For practical reasons one might want to work with a finite subset of behaviors. How does one choose such a finite subset of behaviors ?

(iii) As there may be more than one plan that steers the robot to the goal one might be interested in formulating some notions of performance (optimality) to plans ?

**Proposition 2** *Given an obstacle-free environment and a kinetic state machine that is governed by the differential equation*

$$\dot{x} = \sum_{i=1}^m b_i(x)u_i; \quad x \in \mathbb{R}^n, u \in \mathbb{R}^m \quad (3.4)$$

*such that the control Lie algebra (i.e. the vector space spanned at any point by all the Lie brackets of the vector fields  $b_i$ ) has rank  $n$ , then there exists an alphabet  $\Sigma(\Sigma_e^*)$  which can be used to generate behaviors and hence plans to steer the system from a given initial state  $x_o$  to a final state  $x_f$ .*

**Proof:** (i) From Chow's theorem we know that if the control Lie Algebra has rank  $n$  then the system is controllable. This implies there exist controls  $u : [0, T] \rightarrow \mathbb{R}^m, T \geq 0$  that steer the system from any initial state  $x_o(0)$  to any final state  $x_f(T)$ .

A simple alphabet that can be used to generate behaviors consists of  $m$  triples of the form  $(U_1, 1, 1), \dots, (U_m, 1, 1)$   $\alpha \in \mathbb{R}, \beta \in \mathbb{R}^+$  where

$$\begin{aligned} U_1 &= (1, 0, 0, 0, \dots, 0) \\ U_2 &= (0, 1, 0, 0, \dots, 0) \\ &\vdots \\ U_m &= (0, 0, 0, \dots, 0, 1) \end{aligned} \quad \square$$

**Example 1.** Consider the problem of path planning with a unicycle, with a single sensor, that wanders around in a given environment without colliding

into obstacles (analogous to the idea of the zeroth level of competence in Brooks [6] ). Let us assume the task of the robot (unicycle) in this case is to wander till it senses an obstacle. If it senses an obstacle it avoids the obstacle and continues to wander around. We now formulate and solve this problem treating the unicycle with its sensor as a kinetic state machine and find a plan that solves the problem. The differential equations governing the kinetic state machine are

$$\dot{x} = v_1 \cos \theta \quad (3.5)$$

$$\dot{y} = v_1 \sin \theta \quad (3.6)$$

$$\dot{\theta} = v_2 \quad (3.7)$$

where  $(x, y) \in \mathbb{R}^2$  denotes the position of the unicycle w.r.t some inertial frame,  $\theta \in S^1$  denotes the orientation of the unicycle relative to the horizontal axis,  $v_1$  and  $v_2$ , the velocity of the unicycle and the angular velocity respectively are the inputs to the kinetic state machine . To solve this problem let us consider the following atoms:

$\sigma_1 = (U_1, \xi_1, T_1)$  where

$$\begin{aligned} U_1 &= (1, 0) \\ \xi_1 &= \begin{cases} 1 & \text{if } \rho > 10 \\ 0 & \text{if } \rho \leq 10 \end{cases} \\ T_1 &\in (0, \infty) \end{aligned}$$

where  $\rho$  is the distance between the robot and the obstacle that is returned by the sensor.

$\sigma_2 = (U_2, \xi_2, T_2)$  where

$$U_2 = (0, 1)$$

$$\xi_2 = \begin{cases} 0 & \text{if } \rho > 10 \\ 1 & \text{if } \rho \leq 10 \end{cases}$$

$$T_2 \in (0, \infty)$$

$\sigma_3 = (U_3, \xi_3, T_3)$  where

$$U_3 = (0, 1)$$

$$\xi_3 = 1$$

$$T_3 \in (0, \infty)$$

Let  $\alpha \in [\alpha_{min}, \alpha_{max}]$  and  $\beta \in [0, \infty]$ .

now consider the following behaviors

$$\pi_1 = (\alpha_1^1, \beta_1^1)(U_1, \xi_1, 1)$$

$$\pi_2 = (\alpha_1^2, \alpha_1^2)(U_2, \xi_2, 1)$$

$$\pi_3 = (\alpha_1^3, \beta_1^3)(U_3, \xi_3, 1)$$

Based on the equations of this robot, the behavior  $\pi_1$  is interpreted as “move forward” with a velocity of  $\alpha_1^1$  units/sec for  $\beta_1^1$  seconds and behaviors  $\pi_2$  and  $\pi_3$  can be interpreted as “turn” with a velocity of  $\alpha_1^i$  deg/sec for maximum

of  $\beta_1^i$  seconds i.e., turn right by a maximum of  $\beta$  degrees, unless interrupted. As explained earlier the atoms of each behavior will only execute as long as their respective  $\xi$  functions are 1 and the time of execution is less than  $T$ .

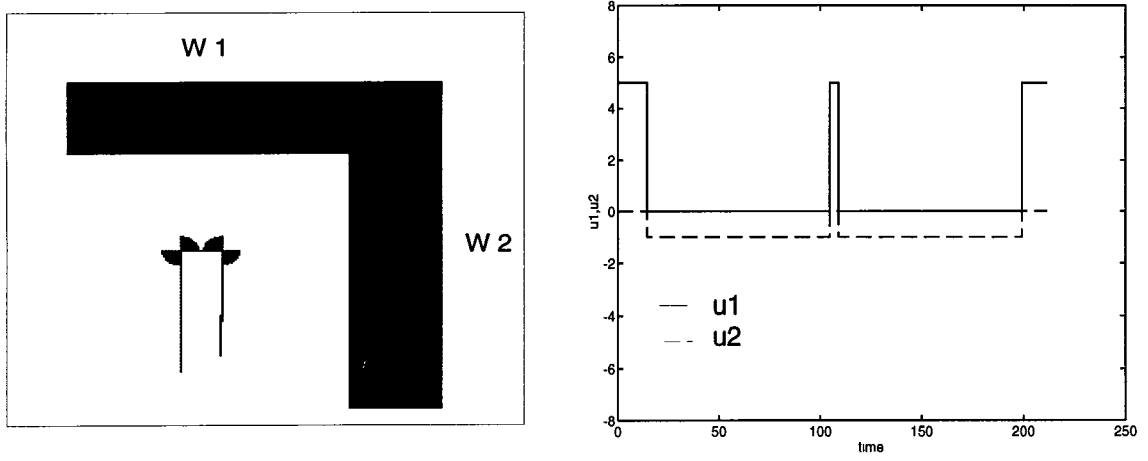


Figure 3.1: Trajectory and Inputs Generated by the Plan

Consider the following plan  $\Gamma = ((5, 100)\pi_1(-1, 90)\pi_3)^*$  i.e.

$$\Gamma = \{(\alpha_1^1, \beta_1^1)\pi_1 (\alpha_1^3, \beta_1^3)\pi_3 (\alpha_1^1, \beta_1^1)\pi_1 (\alpha_1^3, \beta_1^3)\pi_3 \dots\}$$

If this plan is executed in the environment as shown in the Fig. 3.1 observe that the robot will move forward for time  $\hat{t}$ ,  $\hat{t} \leq 100$  when  $\xi_1$  will interrupt it. The execution of behavior  $\pi_1$  is inhibited, behavior  $\pi_3$  is picked up from the queue and is executed. As  $\xi_3 = 1$  in the entire interval  $t \in [\hat{t}, 90]$  the robot will then turn clockwise by 90 degrees and then it will move forward (execute behavior  $\pi_1$ ). But again after some finite time wall W2 (see Fig. 3.1) will

cause  $\xi_1 = 0$  and hence interrupt the move forward behaviour. Behavior  $\pi_3$  is executed as earlier i.e. the robot turns clockwise by 90 degrees, and now continues to move forward. If it does not detect an obstacle at the end of 100 seconds since it started moving forward, it will stop turn clockwise by 90 degrees and continue to repeat its behavior.

Now consider the plan  $\Gamma = ((50, 2)\pi_1(-20, 5)\pi_2)^*$ . If this plan is executed in the same environment (see Fig. 3.2) observe that while executing the “move forward’ i.e.  $\pi_1$ , in the time interval  $0 \leq t \leq 2$  the robot realizes that the obstacle is at a distance less than 10 units from it and hence  $\xi_1$  interrupts the “move forward ” and the robot begins to execute “turn right”. Due to the choice of the interrupt function  $\xi_2$  robot will now switch between “turn right” and and “move forward” (a condition reffered to as chattering) and trace a trajectory as shown in the figure. Hence depending on the choice of the alphabet one can generate different plans to achieve the same task.

The question of how to generate a plan given an alphabet and a kinetic state machine, is an open one and it largely depends on the task. In the next chapter we describe a path planner for nonholonomic robots. Before we discuss the features of the planner we introduce some more definitions that help formalize measures to evaluate the performance of a plan.

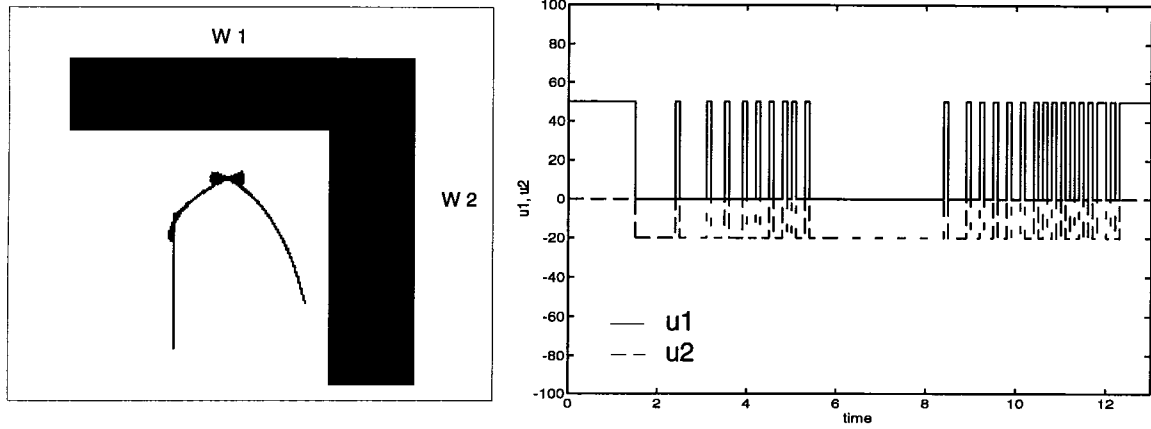


Figure 3.2: Trajectory and Inputs Generated by the Plan

### 3.2 Performance Measure of a Plan

To generate a plan to steer a system from a give initial state  $x_0$  to a final state  $x_f$  requires complete *a priori* information of the world, which is not always the case in most instances of path planning. If we assume that the system does not have complete *a priori* information about the world  $\mathcal{W}$  then the planning system has to generate a sequence of plans based on the limited information about  $\mathcal{W}$  that it has which when concatenated will achieve the required goal. We call these plans that are generated on limited information to achieve some subgoal as *Partial Plans*  $\Gamma^p$ . The plan to steer the system from a given initial state  $x_o$  to a final state  $x_f$  is then determined after the system has reached the final state and is:  $\Gamma = \Gamma_1^p \Gamma_2^p \cdots \Gamma_n^p$  where  $\Gamma_i^p$  is the partial plan consisting of only those behaviors that have been executed for

$t > 0$ .

**Remark:** As the partial plan is generated with limited information of the world, all the behaviors generated by the partial plan may not be executed in real time. For example let us consider a partial plan  $\Gamma^p = \{\pi_3\pi_1\pi_4 \cdots \pi_n\}$ . Let us assume that the behavior  $\pi_1$  is interrupted by  $\xi_1$  at  $\hat{t}$ . Now as by the definition of the plan behavior  $\pi_4$  will begin to execute. But if  $\xi_1 = \xi_4$  the behavior  $\pi_4$  will not be executed.

The length of a plan is given by  $|\Gamma| = \sum_{i=1}^n |\pi_i|$  and the time of execution of the plan is given by  $T(\Gamma) = \sum_{i=1}^n T(\pi_i)$ .

Now that we have these formal definitions we can start defining the performance of an algorithm that uses these behaviors and analyze some earlier algorithms for nonholonomic motion planning.

Given an algorithm that generates a plan  $\Gamma$  we define a measure of performance  $\Theta(\Gamma)$  of the plan as

$$\Theta(\Gamma) = T(\Gamma) + \tau|\Gamma| \tag{3.8}$$

where  $\tau$  is a normalizing factor having the units of time.

Observe that the performance of a plan implicitly depends on the kinetic state machine, which determines the choice of the behavior  $\pi$ .

Defining a performance measure for a path planner is a rather difficult task as it is largely dependent on the goal the robot has to achieve. Some path

planners use the total time to achieve the goal as a measure of the performance of the system. In many situations one might be interested in not only the time but also on the smoothness of the path traversed or the number of times switching between different controls was necessary. For example consider the task of parallel parking of a car. One might be able to achieve the goal by using only open loop controls but switching between them at regular intervals, hence possibly reducing the time to achieve the goal but compromising on the smoothness of the path. On the other hand if one uses a time dependent feedback law, the same task could be possibly achieved by moving along a smooth trajectory but this time taking a longer time to achieve the goal. This indicates a trade off between the two strategies which is captured by the performance measure defined by (3.8).

We now define the optimal performance of a plan as

$$\Theta(\Gamma)_{optimal} = \min\{T(\Gamma) + \tau|\Gamma|\}. \quad (3.9)$$

Here the minimization is performed over the subset of plans defined by the subset of behaviors. Depending on the kinetic state machine and the choice of the planner one can now place bounds on the optimal performance and hence compare the performance of different planners given the same language or that of the planner given a new language. This is illustrated in the example given below.

**Example 2:** Consider the problem of steering the unicycle from a given initial location  $z_o$  to  $z_f$ . The equations of the unicycle are given in example 1. Let us assume that the language consist of the following atoms.  $\sigma_1 = (U_1, \xi_1, 1)$ ,  $\sigma_2 = (U_2, \xi_2, 1)$  where  $U_1, \xi_1, U_2, \xi_2$  are as defined in example 1. Let  $\alpha \in [-5, +5]$  and  $\beta \in (0, \infty)$

Let us also assume that the planner did not have complete information about the world and had to generate  $n$  partial plans to achieve the goal. Each partial plan consists of steering the unicycle from  $z_i$  to  $z_j$  (see Fig 3.3) such that there are no obstacles in some small neighborhood of the the line segment joining these two locations. Let us further make an assumption that the planner uses  $\alpha_i \in [1, -1]$  as the scaling factor while generating partial plans.

From the kinematic equations of the unicycle we know that a simple partial plan to steer a unicycle from  $z_i = (x_i, y_i, \theta_i)$  to  $z_j = (x_j, y_j, \theta_j)$  would be :

- (i) turn by  $(\theta_{z_i z_j} - \theta_i)$ ,
- (ii) move by a distance  $d_i$  and
- (iii) finally turn by  $(\theta_f - \theta_{z_i z_j})$ ,

where  $z_i z_j$  is the vector joining  $z_i$  and  $z_j$ ,  $d_i = \|z_i z_j\|_2$  and  $\theta_{z_i z_j}$  is the orientation of the vector w.r.t. to the x-axis.

We can rewrite this simple algorithm as a partial plan derived from the

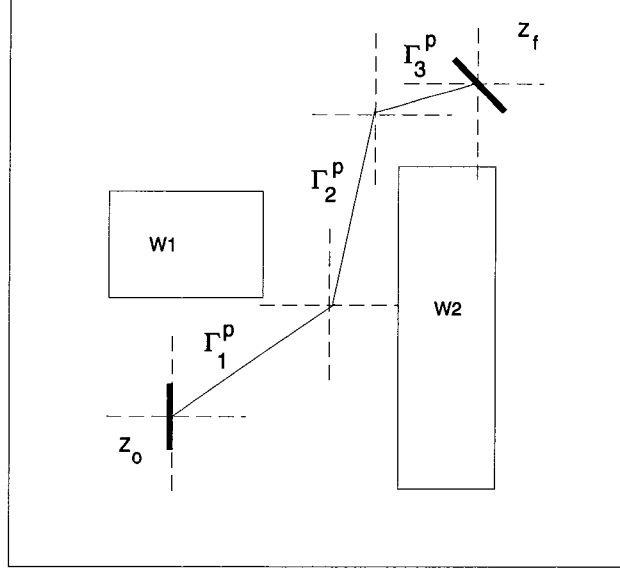


Figure 3.3: Partial Plan Generation

language using the behaviors  $\pi_1 = \sigma_1$  and  $\pi_2 = \sigma_2$  .

$$\Gamma_i^p = \{(\theta_{i1}/abs(\theta_{i1}), \theta_{i1})(\sigma_2), (1, d_i)(\sigma_1), (\theta_{i2}/abs(\theta_{i2}), \theta_{i2})(\sigma_2)\}$$

where  $\theta_{i1}$  and  $\theta_{i2}$  are the angles of the two turns as described above of the  $i$ th partial plan. Hence the plan to steer the system from  $z_o$  to  $z_f$  is given by

$$\Gamma = \{\Gamma_1^p \Gamma_2^p \dots \Gamma_n^p\}$$

Given a plan we now illustrate how bounds can be placed on the optimal performance based on the knowledge of the kinetic state machine and the language.

Let  $d_{max} = \max \|x_i x_j\|$ . But since the planner uses  $\alpha_i \in [-1, 1]$

$$T(\Gamma_i^p)_{max} \leq 2\pi + d_{max} + 2\pi$$

$$= 4\pi + d_{max}$$

Since

$$|\Gamma_i^p|_{max} \leq 3$$

$$0 \leq \Theta(\Gamma) \leq 3n + n(4\pi + d_{max})$$

But since we are using only open loop controls we know from the kinematics of the system that given an initial state  $x_o$  and a final state  $x_f$  both the behaviors  $(\alpha_i, \beta_i)\sigma_i$  and  $(k\alpha_i, \beta_i/k)\sigma_i$  would steer the kinetic state machine from the initial state to the final state, we could replace  $(\alpha_i, \beta_i)\sigma_i$  by  $(k\alpha_i, \beta_i/k)\sigma_i$ .

Hence from our assumption that the planner uses only  $\alpha_i \in [-1, 1]$  and observing that if in our language  $\alpha_i \in [-5, 5]$  it implies that

$$0 \leq \Theta(\Gamma)_{optimal} \leq \frac{n(4\pi + d_{max})}{5} + 3n$$

Having placed bounds on a plan generated by one set of behaviors we can now compare the performance of another set of behaviors (may be one using periodic functions to steer the robot) against these bounds.

## **Chapter 4**

# **Path Planning and Obstacle**

## **Avoidance**

We have seen in the earlier chapters that path planning for nonholonomic systems with (time dependent) state feedback involves both open loop and closed-loop controls. This motivates the need for an intelligent trade off between closed loop and open loop control for tasks of path planning and obstacle avoidance. In chapter 3 we developed a formal language for planning which integrates features of modern control theory and behavior based planning as discussed in chapter 3. In this chapter we present an architecture for real time implementation of the language and a general purpose algorithm for obstacle avoidance with nonholonomic robots. The specific details of the algorithm are described using a robot that is designed along the lines of a unicycle and/or a car with a front wheel used for steering and driving purposes. The equa-

tions of motion and closed loop feedback laws that were described in chapter 2 are used where explanation using an example is required for a better understanding. The algorithm makes certain assumptions on the world, and certain pragmatic assumptions about the sensing capabilities of the robot which are consistent with many real-time applications using mobile robots. In the rest of this chapter, plans, partial plans, behaviors and atoms are used in the sense as defined in chapter 3.

## 4.1 Control Strategy

As we seek to incorporate higher levels of autonomy in robots, the need for hierarchical and distributed control schemes that have a biological analog becomes apparent. Based on our current knowledge of the organization of the mamalian motor systems we might consider segments of the limbs as kinetic state machines, with motor commands to the muscles and tendons and sensor information (that enters into a low level feedback at the spinal level) as inputs to the machine. Motivated by the hierarchical structure of neuromuscular control [23] and observing that the low level spinal reflex control runs faster (loop delays of about 30ms) than the high level feedback loop (100-200 ms delays), we present a control scheme (see Fig. 4.1), to generate and execute plans to achieve a given task.

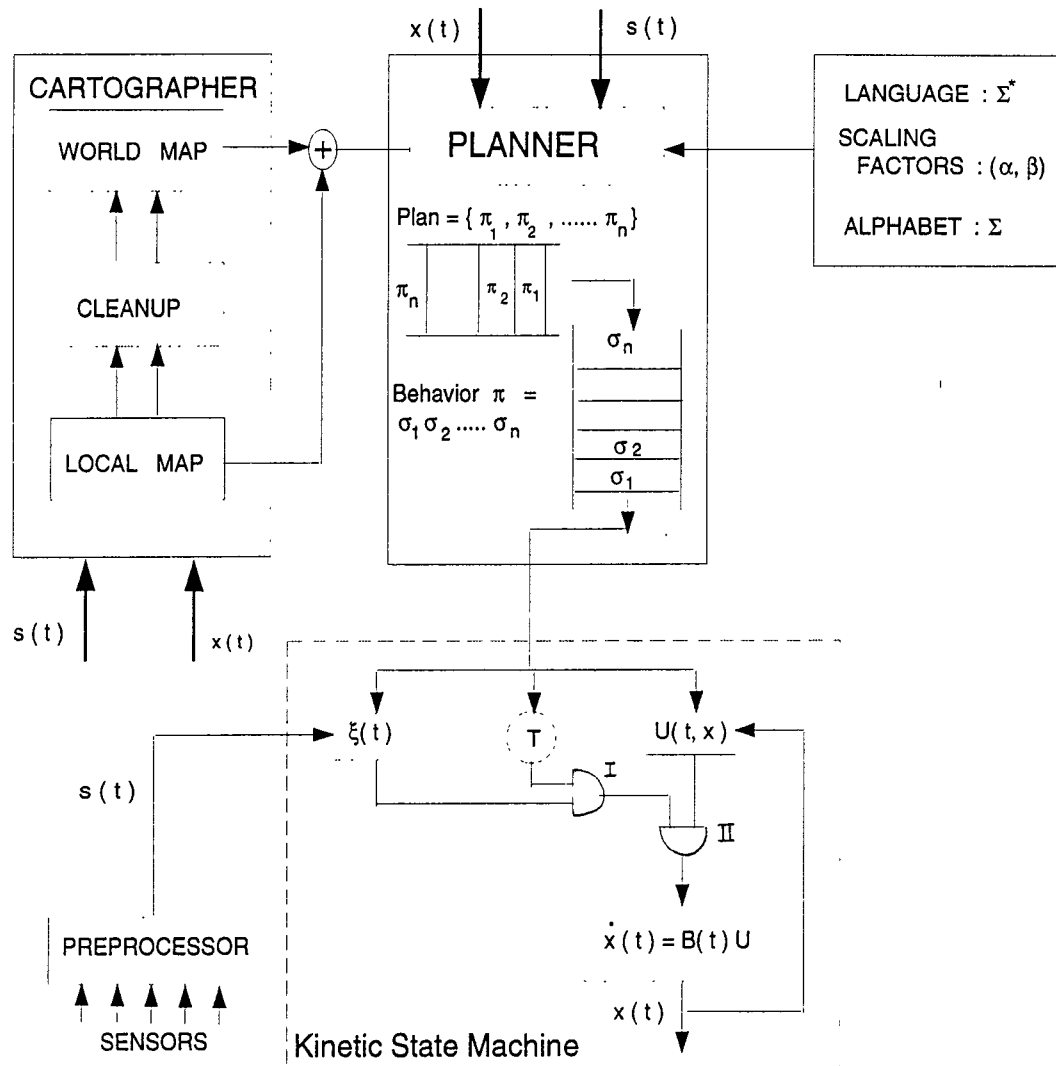


Figure 4.1: Hybrid Control Architecture

The lowest level (the kinetic state machine with sensors and  $(U, \xi, T)$  inputs) could be interpreted as a low level feedback controller operating at the spinal level. The planner could be interpreted as the higher end of the nervous system (cortex + memory..) where sensory information has been processed to generate goal related trajectory information. The distributed nature of the control becomes apparent when one observes that once a plan has been generated each level and even various modules at the same level (e.g. cleanup and plan in Fig 4.1) continue to execute independently. As explained in chapter 3, if the kinetic state machine receives an input  $(U_i, \xi_i, T_i)$  it evolves according to equation 3.2. Interpret  $T_i$  as a timer whose output is 1 (active high) while  $t < T_i$  and is 0 (active low) if  $t \geq T_i$ . As explained earlier  $\xi(s(t))$  is a boolean function that returns an interrupt (active low) to the system when conditions defined by  $\xi(s(t))$  are satisfied. Hence the functioning of the AND gates in the kinetic state machine can be interpreted as follows - if either the robot receives an interrupt or  $t \geq T_i$ , the input to gate II is an active low and hence the input to the kinetic state machine is inhibited i.e. the current behavior is stopped and the next behaviour in the queue is executed.

It is assumed that the robot has no knowledge about the shapes of the obstacles. Planning is restricted to the two dimensional cartesian space. The plan generated consists of an ordered sequence of partial plans each of which is generated based on the limited local information that the robot has. As

obstacles are encountered *en route* to the goal the world map is updated, and a new path to the goal is again planned using the new information now available to the planner. As the partial plans are generated based on local information the paths initially generated are locally optimal (in the sense of chapter 3). As the knowledge of the environment increases the performance of the system begins to improve. Inaccuracies in the plan due to incorrect sensor information do not pose any problem because sensor information is present in the low level feedback loop at run time i.e. while the plan is being executed. The environment is represented as a network of free-space regions.

## 4.2 Navigation Task Composition

The task of navigation and obstacle avoidance can be decomposed into a number of subtasks each being executed at a certain level in the control architecture. Refer to Fig. 4.2 for a diagram of task decomposition.

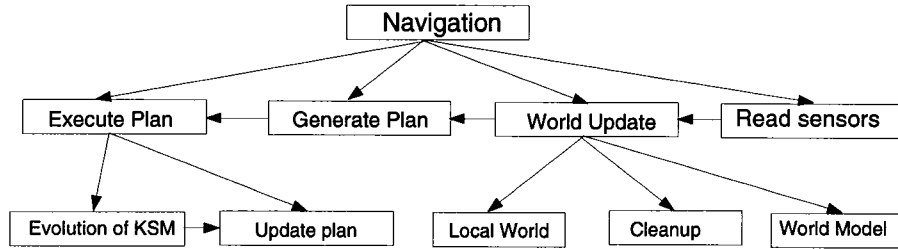


Figure 4.2: Navigation Task Decomposition

**Read sensors** involves reading the distance information returned by the

sensors and determining the the largest obstacle free disk centered around the robot such that the trajectories that lie in this disk are guaranteed not to intersect with the boundaries of the obstacle. In the implementation of the control strategy the sensors are read continuously and distance information and  $\rho_{ofd}$  (the radius of the obstacle free disk) are updated into a global variable which is used to update the world and is also used by **execute plan** as an input to the interrupt function  $\xi$ .

Since the environment is represented as a network of free-space regions, **update world** involves periodic addition of nodes to the network. In the actual implementation of the planner we differentiate between the local world and the global world. The local world is represented as a linked list of intersecting obstacle-free disks, each of which is added to the list in the order they were visited over some finite interval of time. Each node (see Fig. 4.3) contains information regarding the sensor distance information and  $\rho_{ofd}$ . Partial

```

Struct legal_highway
{
    float sensor[10],    /* array with distance information of each IR */
        rho_ofd,        /* radius of obstacle free disk */
        center [2],     /* x, y, coordinates of the obstacle free disk */

    struct legal_highway * next_ofd,
                          * prev_ofd;
}

```

Figure 4.3: Representation of the World

plans use this information regarding the obstacle-free disks to generate be-

haviors that steer the robot to the boundary of the disk. After the complete or partial execution of the partial plan, the robot generates the next partial plan based on the sensor information available at that instant. Note that even though the sensors are continuously being monitored, a new node is added to the list only when a new partial plan needs to be generated. Hence as shown in Fig. 4.4 the world is updated at time instances corresponding to the points A, B, C and so on.

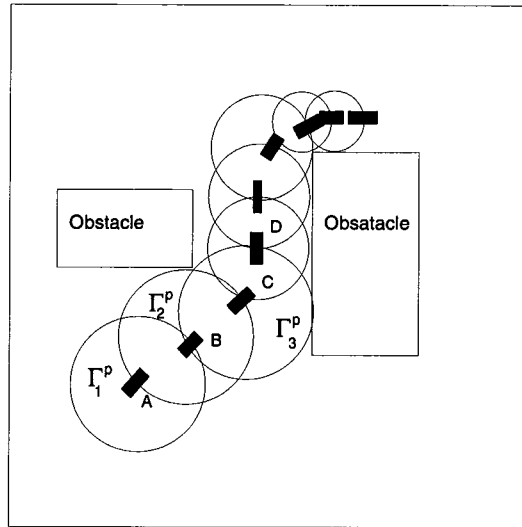


Figure 4.4: World Update

**Generate Plan/Partial Plan** - This involves the interpretation of the sensor information, and language to generate a sequence of behaviors that will steer the robot from its current location to a desired location. If a partial plan is being generated the desired location lies on the boundary of the obstacle free disk. The generation of plans/partial plans is discussed in further detail

in section 4.3.

In the implementation of the planner, the data structure of a plan (see Fig. 4.5) is a linked list of atoms, each atom being represented by a structure that has information regarding the scaling factors  $(\alpha, \beta)$ , the interrupt function  $\xi$ , the maximum time of execution of the kinetic state machine, the inputs (controls) and a pointer to the kinetic state machine. As mentioned earlier it is possible that while executing a partial plan only some of the behaviors may be executed as planned, some may be executed only for a fraction of the the intended execution time and others may not be executed at all. **Update Plan** updates the fields of each atom of the partial plan after it has completed its execution.

```
Struct plan
{
    float max_exec_time, /* maximum time of execution */
          act_exec_time, /* actual time for which atom was executed */
          alpha, beta,   /* scaling factors associated with the atom */
          * ( * control ) ( ); /* Input U to the differential equation */

    int * ( interrupt ) ( ); /* interrupt function associated with the atom */

    void * ( KSM ) ( ); /* equations of motions of the robot */

    struct plan * next_plan,
                * prev_plan;
}
```

Figure 4.5: Representation of a Plan

**Execute Plan** involves decomposing these plans into atoms and letting

the kinetic state machine evolve accordingly. It involves a continuous scan of the sensor information as  $\xi(s(t))$  requires run time sensor information to inhibit the evolution of the kinetic state machine.

As the local world is a list of nodes, each of which is added to the list as the robot generates partial plans it is possible that while generating a plan the robot may have to revisit a node, and hence introducing redundant information into the world. **Cleanup** keeps track of these redundancies in the map and deletes/adds nodes in the list. It also uses the local map information available and builds a graph model of the world map which can be later searched using standard algorithms like  $A^*$  to generate optimal paths to nodes on the graph. This module is currently in its implementation stages.

In the rest of the discussion we restrict ourselves to obstacle avoidance and path planning with a unicycle and/or a front wheel drive cart, equipped with a finite number of infrared sensors that return distance information.

### 4.3 Partial Plan Generation

As mentioned earlier a partial plan should consist of behaviors that steer the robot in these obstacle free disks such that the trajectories do not leave the disk, hence avoiding obstacles, and the direction of travel should be such that the net effect of the partial plan is to steer the robot towards the goal in a

global sense. We would also like to generate plans that have nearly optimal performance, where the performance measure is as defined in chapter 2. Hence assuming that for the given language there exists more than one behavior to steer the robot to the given subgoal, the planner now has an additional task of choosing the behavior that yields a close to optimal performance. This could be done either by generating all possible partial plans and then choosing the one with an optimal performance or making the decision based on a set of rules which depend on the interpretation of the environment. For example, it is intuitively clear that to steer a car in an obstacle-free disk of small radius using pure open loop controls would result in a significantly large number of switching between controls and hence resulting in a path of high complexity. In this situation one could view complexity as the number of turns the robot has to make. Before we go into the actual details of plan generation we introduce some definitions and assumptions that we make about the robot and the world.

### 4.3.1 Assumptions and Definitions

**Assumptions:**

- (i) At any given time the robot has information about its current coordinates and the coordinates of the goal. It has no *a priori* information on the location

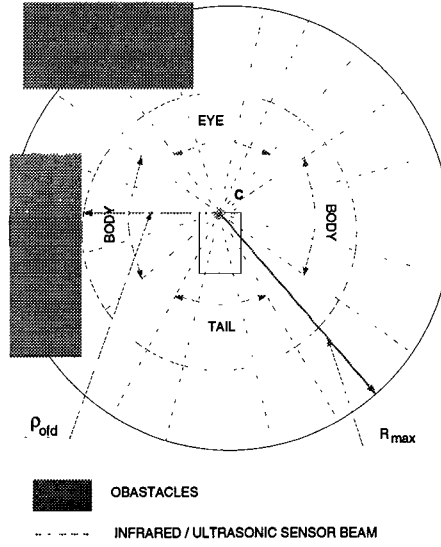


Figure 4.6: Location and Calibration of IR sensors

and shape of the obstacles. No constraints are imposed on the size and shape of the obstacles.

(ii) Location and calibration of sensors on the robots: For purposes of obstacle detection, ranging and mapping, the robots are provided with infrared/ultrasonic sensors located as shown in the Fig 4.6. Bump sensors are located along the body of the robot. Optical encoders mounted on the motors provide position and velocity feedback. The infrared sensors are classified as  $S^i_{eye}$  if they are located in the eye, as  $S^i_{body}$  if they are located along the sides or as  $S^i_{tail}$  if they are located at the rear of the robot (see Fig. 4.6). The sensors return distance information and we shall refer to the distance information returned as  $\rho^i_{eye}$  or  $\rho^i_{body}$  depending on which sensor detects an object along its line

of sight. The sensors are calibrated such that the maximum range of each of these sensors is on the boundary of a circle of radius  $R_{max}$  centered at C. The sensors located at the rear of the robot are normally turned off and are turned on only when the robot is reversing. While reversing they function in a manner similar to those of the eye.

**Definition :** Obstacle free disk  $B_{OFD}(C, \rho_{ofd})$  is defined as a disk of radius  $\rho_{ofd}$  ( $\rho_{ofd} \leq R_{max}$ ) centered at C (see Fig. 4.6) such that there are no obstacles in this ball, i.e.  $\rho_{ofd} = \min[\rho_{eye}^i, \rho_{body}^j]$ ,  $i = 1, \dots, n$ .  $j = 1 \dots n$ . Any trajectory that lies within this ball does not intersect the boundaries of any obstacle

As the distance between the robot and the obstacles decreases, due to the nonholonomic constraints steering might become more difficult and the planner might need to adopt a different control scheme to steer the robot. Let  $R_{crt} \leq R_{max}$  denote the radius of the ball centered at C such that the planner needs to use a different set of controls to steer the robot from  $x_o$  to  $x_f \in \partial \rho_{ofd}$ .

In the actual implementation of the planner we place a threshold on the radius of the obstacle free disk and have a rule according to which if  $\rho_{ofd} \leq R_{crt}$  then we use closed loop controls to steer the robot. Hence the problem of generating partial plans reduces to (i) planning in an obstacle free disk with  $\rho_{ofd} > R_{crt}$  and (ii) planning in an obstacle free disk such that  $\rho_{ofd} \leq R_{crt}$ . Each of the cases is discussed in the following subsections.

### 4.3.2 Planning in the Obstacle Free Disk

To find the best direction of travel in this case we use the approach of potential functions. As in the earlier work on path planning with potential functions the idea behind our construction is based on electrostatic field theory - charges of the same sign repel and charges of the opposite sign attract in accordance with Coulomb's law. Hence we assign a positive charge distribution to the obstacles and the mobile robot and a negative charge distribution to the goal and assign an artificial potential function to the robot. We use the resultant gradient field only to determine the scaling factors and  $x_f \in \partial\rho_{ofd}$ , the circumference of the obstacle free disk, as the integral curves may not result in feasible trajectories. The idea is to construct a vector field which will give the best direction of travel based on the location of the obstacles and the goal. The robot is approximated by a point robot and sensors can detect only points on the boundaries of the obstacles that lie in their line of vision, we treat obstacles as point charges and assign charges to them depending on which sensor detects them. The sum total of both the attractive and repulsive forces is used to determine the bounds on the velocities and hence the bounds on the scaling factors. We also assign weights to the forces due to the goal

Let  $\lambda_{eye}^i$ ,  $\lambda_{body}^j$  be charges associated with points on the boundaries of the obstacle/goal detected by the sensors on the robot. Depending on which

sensor detects them we assign them values as follows:

- (i)  $\lambda^j_{body} > 0$  if  $\rho^j_{body} < R_{max}$
- (ii)  $\lambda^i_{eye} > \lambda^j_{body} > 0$  if  $\rho^i_{eye} < R_{max}$

Observe that the length of the shortest feasible path to move to a configuration that lies in the cone of the eye is less than the length of the shortest feasible path to a configuration that lies at the same euclidean distance along (near) the body. The choice of  $\lambda^i_{eye} > \lambda^j_{body}$  ensures that if an obstacle is detected at the same euclidean distance by  $S^i_{eye}$  and  $S^j_{body}$  the change in trajectory required due to an obstacle detected at a distance  $\rho^i_{eye}$  is more than that required if an obstacle is detected at  $\rho^j_{body}$ .

We associate subgoals (negative charges) when no obstacle is detected by a sensor in the eye to suggest a possible free path in the direction of motion.

- (iii)  $\lambda^i_{eye} < 0$  if  $\rho^i_{eye} = R_{max}$
- (iv) The gradient field associated with the obstacles is given by

$$\nabla f_{obs} = \left( \sum_{i=1}^n \lambda^i_{eye} \nabla f_i^{eye} + \sum_{i=1}^n \lambda^j_{body} \nabla f_i^{body} \right)$$

The gradient field associated with the goal is

$$\nabla f_g = \|\nabla f_{obs}\| u_g$$

$u_g = \frac{(X_{robot} - X_{goal})}{\|(X_{robot} - X_{goal})\|}$  where  $X_{robot}$  and  $X_{goal} \in \mathbb{R}^2$  are the coordinates of the (point) robot and the goal respectively. where

The resultant gradient field is given by:

$$F = \left( \sum_{i=1}^n \lambda_{eye}^i \nabla f_i^{eye} + \sum_{i=1}^n \lambda_{body}^j \nabla f_i^{body} \right) + \nabla f_g$$

and the best direction of travel is given by the  $\frac{F}{\|F\|}$ . The intersection of this vector with the boundary of the obstacle free disk determines the final location to which the robot has to be steered.

Observe that in the obstacle free disk the resultant gradient field has only one equilibrium point, the center of the obstacle free disk and it occurs when

$$\left\langle \frac{\nabla f_{obs}}{\|f_{obs}\|}, u_g \right\rangle = -1$$

In such a situation we disturb the location of the goal by a small amount. Having calculated the location on the boundary of the disk to which the robot has to be steered we use control laws similar to those discussed in chapter 2 to steer the robot.

### 4.3.3 Tracing Boundaries of Obstacles

Planning in  $B(C, R_{crt})$  is now a closed loop planning strategy which essentially results in a *trace* behavior that traces the boundaries of the obstacles. An example of the closed loop control is given in chapter 2. Given the limited sensor and world information it is probable that the direction of trace may

have been wrong. Hence we use a heuristic function  $f(x) = D(X_{robot}, X_{goal})$ , the euclidean distance between  $X_{robot}$  and  $X_{goal}$  as an estimate of how far the robot has strayed from the goal. Trace the boundary as long as  $f < f_s$  where  $f_s$  is the euclidean distance  $D(X_{robot}, X_{goal})$ , at the instant when the trace behavior was started. If  $f > f_s$  then retrace path and trace the boundary of the obstacle in the opposite direction. If terminal conditions (trace until you find a corner) for trace are not met set  $f_s = 2f_s$  and repeat.

**Remark:** Retracing a path under this framework is a rather simple task. Observing that the system is a drift free system, retracing involves executing the past  $n$  partial plans in a reverse order with  $(-\alpha)$  scaling factor.

## 4.4 Path Execution

Once the plans/partial plans have been generated, these plans have to be executed. The general algorithm for path execution is shown in Figure 4.7. Plan execution involves decomposition of the plan into behaviors and further into atoms. The kinetic state machine is allowed to evolve as explained in Chapter 3, equation 3.2. We would like to mention here that we use two levels of interrupts in the implementation of the path execution module. While a partial plan is being executed if a low level interrupt is received the execution of that particular atom is inhibited and the actual time of execution of the atom is

updated for future use and the next atom in the partial plan is executed. On the other hand if a higher level interrupt is received the execution of the partial plan itself is inhibited and a new partial plan is generated.

Fig. 4.7 and Fig.4.8 show some of the paths generated by the planner using method described in section 4.3.2. It should be pointed out here that the obstacle-free disks generated by the planner violate the definition as stated in section 4.3.2, but this is because in the simulator we have used only sensors of the eye to generate obstacle-free disks. These obstacles that are not detected by the sensors may be thought of as being the blind spots of the robot. It is important to note that while the plan is being executed the sensors are being continuously scanned and are present in a low level feedback loop hence preventing any collisions with obstacles.

## **4.5 Learning and World Model Update Algorithm**

Once the robot has explored the environment using limited range sensors, it is natural to expect the robot to generate plans of a better performance if it has to repeat the same task or move to goal that lie in the explore regions. We suggest a “learning algorithm” that improves the performance of a plan

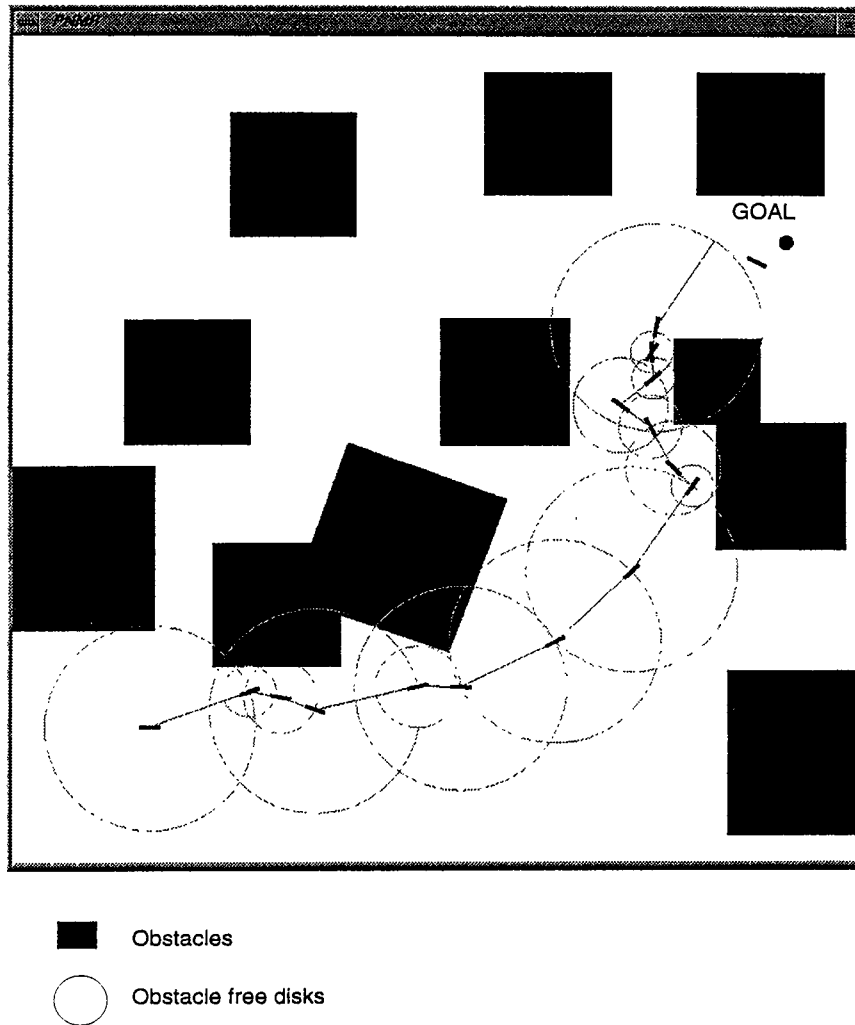
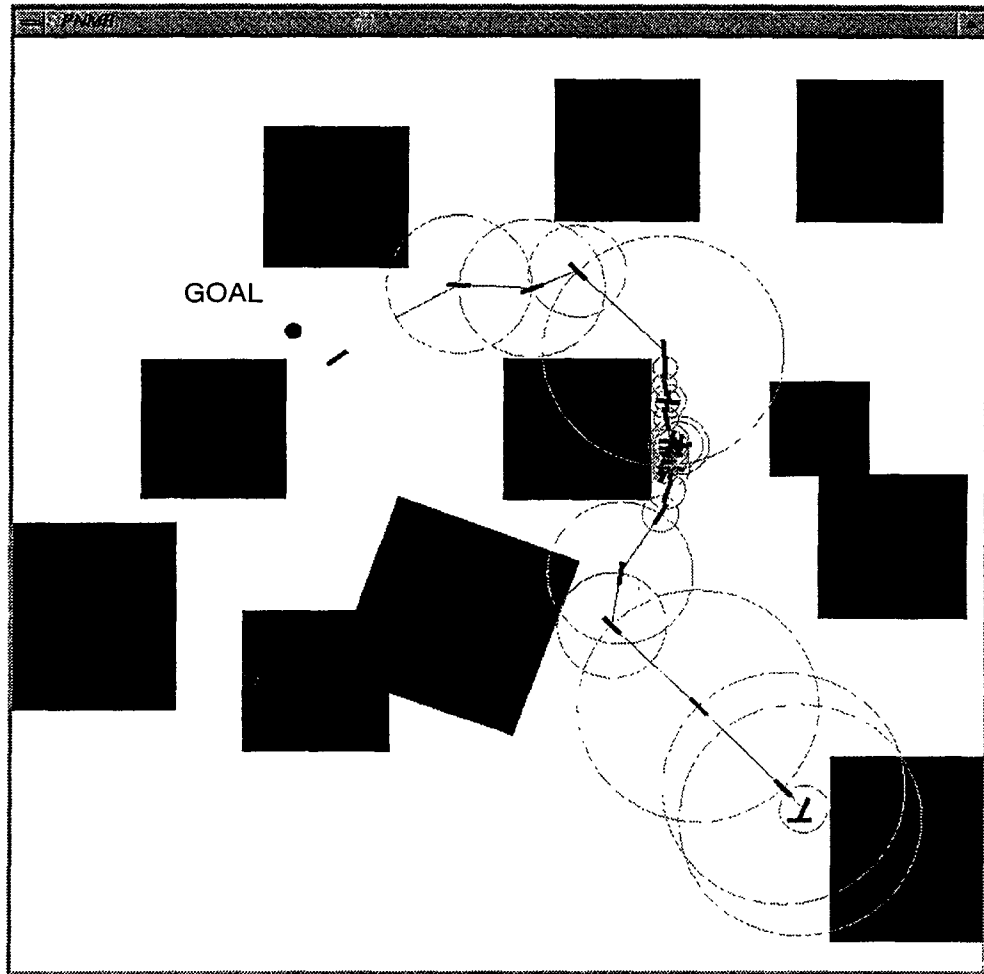


Figure 4.7: Paths Generated by the Planner





-  Obstacles
-  Obstacle free disks

Figure 4.8: Paths Generated by the Planner

to bring it closer to optimal. The main features of the algorithm are discussed below.

As described in section 4.3, the plan to steer a robot consists of a sequence of partial plans, where each partial plan steers the robot in some obstacle free disk of radius  $\rho_{ofd}$ . In the rest of the section let us denote each of the obstacle free disk which were used to generate the  $i$ th partial plan as  $B_i$  and the  $i$ th partial plan as  $\Gamma_i^p$ . Once the plan has been generated the planner makes the following observations. Further let us assume that  $n$  such partial plans were generated.

- (i). If  $B_i \subset B_j$ ,  $i = 1, \dots, n, j = 1, \dots, n$  (i.e.  $B_i$  is contained in  $B_j$ ) then obviously  $B_i$  contains redundant information. Thus if  $B_i \subset B_{i+1}$  the partial plan  $\Gamma_i^p$ , that steers the robot from  $C_i$  to  $C_{i+1}$ , where  $C_i$  is the center of the obstacle free disk, and partial plan  $\Gamma_{i+1}^p$  that steers the robot from  $C_{i+1}$  to  $C_{i+2}$  can be replaced by a partial plan  $\hat{\Gamma}_i^p$  that steers the robot from  $C_i$  to  $C_{i+2}$ . Since  $B_i \subset B_{i+1}$  it is obvious that  $\Theta(\hat{\Gamma}_i^p) < \Theta(\Gamma_i^p \Gamma_{i+1}^p)$
- (ii). Observe that since  $C_{i+1}$  lies on the boundary of  $C_i$ , we are guaranteed the existence of a trivial nonfeasible trajectory (the straight line joining  $C_i$  with  $C_{i+2}$ ) that lies entirely in  $B_i \cup B_{i+1}$  i.e. the obstacle free area enclosed by these two intersecting obstacle free disks. Hence if there exists a partial plan  $\hat{\Gamma}_i^p$  that generates feasible trajectory that can track this nonfeasible trajectory and lie entirely in  $B_i \cup B_{i+1}$  such that  $\Theta(\hat{\Gamma}_i^p) < \Theta(\Gamma_i^p \Gamma_{i+1}^p)$  we can replace

$\Gamma_i^p \Gamma_{i+1}^p$  by  $\hat{\Gamma}_i^p$

(iii) After the execution of (i) and (ii) we now have partial plans that steer the robot from  $C_i$  to  $C_{i+j}$ ,  $j \in \{2, \dots, n\}$  such that the trajectory lies entirely in  $\cup_i^{i+j} B_i$ . The planner now explores the possibility of finding (non)feasible trajectories from  $C_i$  to  $C_{i+j+k}$ ,  $j \in 2, \dots, n, k = 1, \dots, n$  such that these trajectories lie entirely in  $\cup_i^{i+j+k} B_i$  and the performance of the plan that generates this trajectory is better than the earlier one. In Fig. 4.9 we give an example of an algorithm to explore the possibility of a trivial (non)feasible trajectory. Fig. 4.10 and Fig. 4.11 show paths generated by the planner after it has gained partial knowledge of the world it has explored in its first attempt to reach the goal. It clearly shows an improvement in the performance of the planner as the length of the plan is nearly a third of the plan generated in the first attempt.

Remarks: (i). One should note that generating plans of better performance does not necessarily imply that  $|\hat{\Gamma}_i^p| < |\Gamma_i^p|$  where  $\hat{\Gamma}_i^p$  is the new plan but could simply imply choosing the right scaling factors  $\alpha, \beta$  such that  $T(\hat{\Gamma}_i^p) < T(\Gamma_i^p)$ .

(ii). One need not restrict the generation on nonfeasible trajectories to straight line segments, but could instead use arc or even curves that best fit the centers of these obstacle free disks.

```

learn_update ( )
{
    k = 1 ;
    while ( k != n )
    {
        for ( i = n ; i > k ; i -- )
        {
            in_disk_flag = 1 ;

            /* C ( . ) is the coordinates of the center of an obstacle free disk */

             $L_i L_k = \text{eqn\_line} ( C ( i ) , C ( k ) )$  /* equation of the line joining the
                                                    C ( i ) and C ( k ) */
            for ( j = i ; j > k ; --j )
            {
                if (  $L_i L_k \cap B ( C ( j ) ) == 0$  ) /* i.e. if  $L_i L_k$  does not intersect the
                                                            obstacle free ball centered at C( j ) */
                {
                    in_disk_flag = 0 ;
                    break ;
                }
            }

            if( in_disk_flag == 1 )
            {
                generate_partial_plan ( C ( k ) , C ( i ) );
                /* if there exists a path that passes through all the obstacle free
                   disks  $B_k$  to  $B_i$  then generate a partial plan to steer the robot
                   from C ( k ) to C ( i ) */

                break ;
            }
        }

        k = i ;
    }
}

```

Figure 4.9: Learning and Plan Update Algorithm

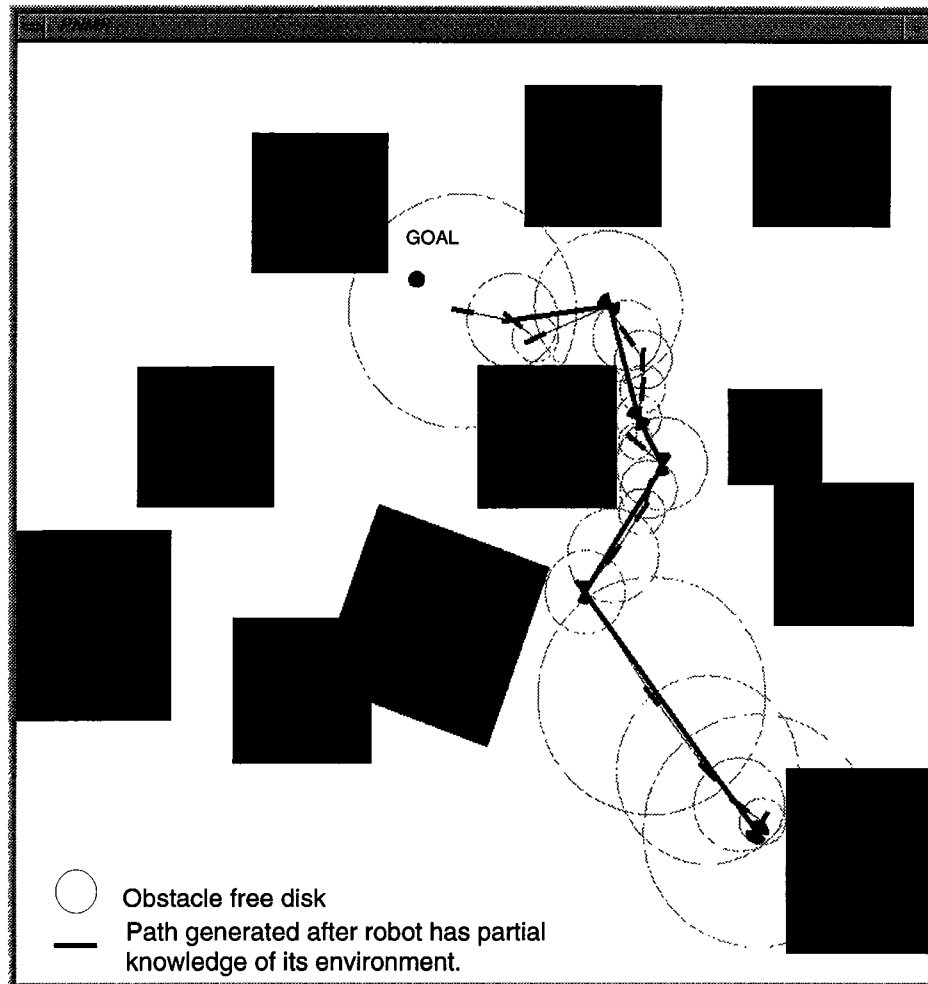


Figure 4.10: Plan Generated after Gaining Partial Knowledge of the World

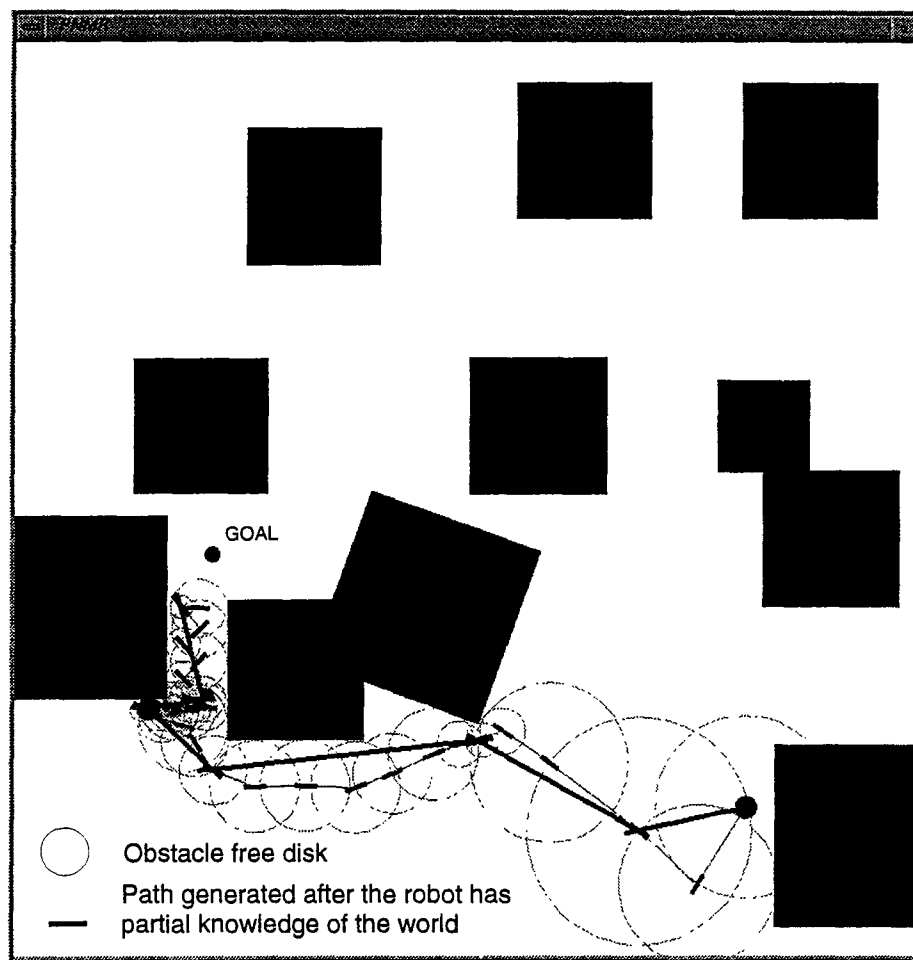


Figure 4.11: Plan Generated after Gaining Partial Knowledge of the World

## Chapter 5

# Conclusions and Future Directions

In this thesis an attempt was made to bring together aspects of motion planning with robots as seen by researchers in the communities of reactive planning and control theory. Some of the conclusions are:

- An investigation into the strengths and drawbacks of the approaches adopted by these two communities to solve the problem of autonomous motion planning suggests that a hybrid strategy that integrates features of both might provide a better solution.
- Since many robotic systems include constraints that are not holonomic, reducing the robot to a point and then designing planning and navigation systems for such robots is not always possible. A complete understanding of the constraints and related issues (controllability, feedback stabilization) is required to design control laws to steer these systems

along feasible trajectories. Depending on the world in which the robot is navigating in many situations it might be helpful to switch from steering using analytical tools based on Lie algebras to planning that relies on the direct coupling of sensory information and actuators.

- To capture features of continuous and discrete time control strategies, an interface is required and this interface is provided via the motion description language suggested in chapter 3. The language serves as a tool to generate control laws or “behaviors” that integrate real time sensor information with continuous time control laws. The language provides a framework to encode and compare various control strategies (piecewise continuous feedback laws, reactive planning) for path planning.
- As we seek higher levels of autonomy in robots the need for a hierarchical and distributed control schemes that have a biological analog becomes apparent. The architecture presented in chapter 4 is in part inspired by some understanding of the mammalian motor system.
- Though steering and stabilization of systems subject to nonholonomic constraints have been studied extensively, many of the steering algorithms and feedback laws suggested cannot be applied for real time applications. To implement control strategies on nonholonomic robots for real time applications the design of controllers that can generate control

sequences that result in “nice trajectories” is essential.

Future research directions should include:

- Continuing the formalization of behavior-based robotics. This should include expanding the concept of the motion description language to include multiple kinetic state machines and communication protocols between kinetic state machines.
- Design of analytical tools based on Lie algebras for real time control of nonholonomic robots.
- Explore the possibilities of using adaptive neural nets to generate intelligent behaviors for path planning with moving obstacles, increasing the levels of parallelism in the control strategy for more efficient real time planning and include error recovery strategies.
- continued development of the simulator for testing the integration of control and reactive techniques in complex (moving and/or movable obstacles) problems.

# Bibliography

- [1] V.J. Lumelsky. Algorithmic and complexity issues of robot motion in an uncertain environment. *Journal of Complexity*, 3:146–182, 1987.
- [2] T. Lozano-Perez. Spatial planning: A configuration space approach. AI memo 605, MIT Artificial Intelligence Laboratory, Cambridge, Mass., 1980.
- [3] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–99, Spring 1986.
- [4] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE International Conference on Robotics and Automation*, pages 1–6, Raleigh, NC, Mar 1987.
- [5] R. Shahidi, M.A. Shayman, and P.S. Krishnaprasad. Mobile robot navigation using potential functions. In *Proceedings of the IEEE Interna-*

*tional Conference on Robotics and Automation*, pages 2047–2053, Sacramento, California, April 1991.

- [6] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
- [7] C.R. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, 1988.
- [8] C.R. Arkin. Behaviour based robot navigation for extended domains. *Adaptive Behaviour*, 1(2):201–225, 1992.
- [9] G.Campion, B.d’Andrea-Novel, and G. Bastin. Controllability and state feedback stabilizability of nonholonomic mechanical systems. In *Lecture Notes in Control and Information Sciences*, pages 107–124. Springer-Verlag, 1990.
- [10] R. M. Murray and S. S. Sastry. Steering nonholonomic systems using sinusoids. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 2097–2101, Honolulu, HI, December 1990.
- [11] H.J. Sussmann. Local controllability and motion planning for some classes of systems without drift. In *Proceedings of the 30th Conference on Decision and Control*, pages 1110–1114, Brighton, England, December 1991. IEEE.

- [19] J-P. Laumond, M. Taix, and P. Jacobs. A motion planner for car-like robots based on a mixed global/local approach. In *IEEE International Workshop on Intelligent Robots and Systems*, Japan, 1990.
- [20] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.
- [21] R.W. Brockett. Formal languages for motion description and map making. In *Robotics*, pages 181–193. American Mathenmatical Society, 1990.
- [22] R. W. Brockett. Hybrid models for motion control. In H. Trentelman and J. C. Willems, editors, *Perspectives in Control*, pages 29–51. Birkhauser Verlag, 1993.
- [23] R.M. Murray, D.C. Deno, K.S.J. Pister, and S.S. Sastry. Control primitives for robot systems. *IEEE Transactions on Systems, Man and Cybernetics*, 22(1):183–193, January/February 1992.